# 目　录

## 第一章　图像及其数字化



## 第二章　图像显示



## 第三章　图像运算

## 第四章 图 像 变 换



## 第五章 滤波和滤波器设计



## 第六章 二值形态学操作

## 第七章 图像的空间变换



## 第八章 图像增强



## 第九章 图像复原



## 第十章 图像编码与压缩

## 第十一章 图像分析和理解



## 第十二章 其他图像处理技术

# 第一章 图像及其数字化

本章要点：

★ 图像及数字图像处理概述
★ 图像处理和分析系统的结构
★ MATLAB 图像处理初步
★ 图像类型

## 1.1 概　　述

### 1.1.1 图像与数字图像

图像就是用各种观测系统以不同形式和手段观测客观世界而获得的，可以直接或间接作用于人眼而产生视知觉的实体。科学研究和统计表明，人类从外界获得的信息约有 75% 来自于视觉系统，也就是说，人类的大部分信息都是从图像中获得的。图像是人们从出生以来体验到的最重要、最丰富、信息量获取最大的部分。

图像能够以各种各样的形式出现，例如，可视的和不可视的，抽象的和实际的，适于计算机处理的和不适于计算机处理的。就其本质来说，可以将图像分为两大类：

一类是模拟图像，包括光学图像、照相图像、电视图像等，例如，在生物医学研究中，人们在显微镜下看到的图像就是一幅光学模拟图像，照片、用线条画的图、绘画也都是模拟图像。模拟图像的处理速度快，但精度和灵活性差，不易查找和判断。

另一类是将连续的模拟图像经过离散化处理后变成计算机能够辨识的点阵图像，称为数字图像。严格的数字图像是一个经过等距离矩形网格采样，对幅度进行等间隔量化的二维函数，因此，数字图像实际上就是被量化的二维采样数组。

本书中涉及到的图像处理都是指数字图像的处理。与模拟图像相比，数字图像具有以下显著优点：

■ 精度高：目前的计算机技术可以将一幅模拟图像数字化为任意的二维数组，即数字图像可以由无限个像素组成，每个像素的亮度可以量化为 12 位(即 4096 个灰度级)，这样的精度使得数字图像与彩色照片的效果相差无几；

■ 处理方便：由于数字图像本质上是一组数据，所以可以用计算机对它进行任意方式的修改，例如，放大、缩小、改变颜色、复制和删除某一部分等；

■ 重复性好：模拟图像(例如，照片)即便是使用非常好的底片和相纸，也会随着时间的流逝而退色、发黄，而数字图像可以存储在光盘中，上百年后再用计算机重现也不会有

丝毫的改变。

### 1.1.2 图像处理技术内容与相关学科

图像处理就是将图像转换为一个数字矩阵存放在计算机中，并采用一定的算法对其进行处理。图像处理的基础是数学，最主要的任务就是各种算法的设计和实现。目前的图像处理技术已经在许多不同的应用领域中得到重视，并取得了巨大的成就。根据应用领域的不同要求，可以将图像处理技术划分为许多分支，其中比较重要的分支有：

■ 图像数字化：通过采样与量化过程将模拟图像变换成便于计算机处理的数字形式。图像在计算机内通常用一个数字矩阵来表示，矩阵中的每一个元素称为像素。图像数字化的设备主要是各种扫描仪与数字化仪；

■ 图像增强与复原：主要目的是增强图像中的有用信息，削弱干扰和噪声，使图像清晰或将其转换为更适合人或机器分析的形式。图像增强并不要求真实地反映原始图像，而图像复原则要求尽量消除或减少在获取图像过程中所产生的某些退化，使图像能够反映原始图像的真实面貌；

■ 图像编码：在满足一定的保真度条件下，对图像信息进行编码，可以压缩图像的信息量，简化图像的表示，从而大大压缩图像描述的数据量，以便于存储和传输；

■ 图像分割与特征提取：图像分割是将图像划分为一些互不重叠的区域，通常用于将分割的对象从背景中分离出来。图像的特征提取包括了形状特征、纹理特征、颜色特征等；

■ 图像分析：对图像中的不同对象进行分割、分类、识别、描述和解释；

■ 图像隐藏：是指媒体信息的相互隐藏，常见的有数字水印和图像的信息伪装等。

上述图像处理的内容往往是相互联系的，一个实用的图像处理系统往往需要结合应用几种图像处理技术才能得到所需要的结果。例如，图像数字化是将一个图像变换为适合计算机处理的形式，这是图像处理的第一步；图像编码技术可用于传输和存储图像；图像增强与复原一般是图像处理的最后目的，当然也可作为进一步进行图像处理工作的准备；通过图像分割得到的图像特征既可以作为最后结果，也可以作为下一步图像分析的基础。

图像处理技术涉及到的知识很广泛，也很复杂。例如，图像的编码理论基础是信息论和抽象数学的结合，进行图像识别需要掌握随机过程和信号处理方面的知识，不少课题还需要更加专业的知识，如小波变换、神经网络、分形理论等。另外，图像处理是一门应用性很强的学问，必须与计算机技术的发展相适应。例如，傅立叶变换是图像处理常用的方法，到目前为止，库利-图基快速傅立叶变换一直是实际应用中的主要算法，该算法需要的乘法数目大大少于一般的傅立叶变换算法，而加法数目则大大增加，因而对于大部分 CPU 来说，总的运算速度提高了很多。但是，Intel 的 CPU 现在加入了 MMX 指令，用该指令计算乘法和加法所用的时间是一样的，所以在 MMX 指令面前，由于加法数目的增加导致快速算法反而显得更慢，因此也就产生了新的适应 MMX 指令的快速算法。

图像处理的另一个特点，也是难点，就是其算法的优劣与被处理对象的内容高度相关，很难找到一种适用于各种情况的通用方法。因此，图像处理按照处理的对象类别又可以分为遥感图像处理、医学图像处理等。另外，以下介绍的学科也和图像处理有着密切的关系：

■ 计算机图形学：用计算机将由概念所表示的物体(不是实物)图像进行处理和显示。

计算机图形学主要是根据给定物体的描述模型、光照及想象中的摄像机的成像几何来生成一幅图像，另外还包括称之为“计算机艺术”的艺术创作。图形和图像本身有着非常密切的关系，因而计算机图形学与图像处理也是相互关联的，但两者的区别也很显著：前者是用点、线、面描述物体，多采用几何手段；后者基本上只和像素点打交道；

■ 模式识别：图像处理的最重要目的之一就是识别，而模式识别技术也是图像技术重要性的体现，诸如指纹鉴别、人脸识别等，都要和模式识别打交道；

■ 人工智能：可以说图像处理、模式识别和人工智能是三位一体的学科。目前模式识别技术遇到的最大困难就是自动化程度不够，即智能程度不高。例如，人眼可以很容易从人群中找到自己要寻找的目标，但对于计算机来说，就连判断图像中是否有人存在这样看似简单的问题都很难解决。神经网络技术给人工智能开辟了一个新的方向，但目前该技术还不够成熟；

■ 计算机视觉：研究计算机视觉的目的是开发出能够理解自然景物的系统。在机器人领域中，计算机视觉能够为机器人提供眼睛的功能，但是这门学科的难度很高。

### 1.1.3 图像处理技术的发展现状

图像处理是人类视觉延续的重要手段，可以使人们看到任意波长上所测得的图像。例如，借助伽马相机、X 光机，人们可以看到红外和超声图像；借助 CT 可看到物体内部的断层图像；借助相应工具可看到立体图像和剖视图像。几十年前，美国在太空探索中拍回了大量月球照片，但是由于种种环境因素的影响，这些照片是非常不清晰的，为此，人们对这些照片应用了一些图像处理手段，使照片中的重要信息得以清晰再现。正是这一方法产生的效果引起了巨大的轰动，从而促进了图像处理技术的蓬勃发展。

总体来说，图像处理技术的发展大致经历了初创期、发展期、普及期和实用化期四个阶段。初创期开始于 20 世纪 60 年代，当时的图像采用像素型光栅进行扫描显示，大多采用中、大型机对其进行处理。在这一时期，由于图像存储成本高，处理设备造价高，因而其应用面很窄。20 世纪 70 年代进入了发展期，开始大量采用中、小型机进行处理，图像处理也逐渐改用光栅扫描显示方式，特别是出现了 CT 和卫星遥感图像，对图像处理技术的发展起到了很好的促进作用。到了 20 世纪 80 年代，图像处理技术进入普及期，此时的微机已经能够担当起图形图像处理的任务。VLSI 的出现更使得处理速度大大提高，其造价也进一步降低，极大地促进了图形图像系统的普及和应用。20 世纪 90 年代是图像技术的实用化时期，图像处理的信息量巨大，对处理速度的要求极高。

21 世纪的图像技术要向高质量化方面发展，主要体现在以下几点：

■ 高分辨率、高速度：图像处理技术发展的最终目标是要实现图像的实时处理，这在移动目标的生成、识别和跟踪上有着重要意义；

■ 立体化：立体化所包括的信息最为完整和丰富，未来采用数字全息技术将有利于达到这个目的；

■ 智能化：其目的是实现图像的智能生成、处理、识别和理解。

### 1.1.4 图像处理技术的应用领域

目前图像处理技术主要的应用领域有：

■ 通讯技术：图像传真，电视电话，卫星通讯，数字电视等；

■ 宇宙探索：其他星体图像的处理；

■ 遥感技术：农林资源调查，作物长势监视，自然灾害监测、预报，地势、地貌以及地质构造测绘，找矿，水文、海洋调查，环境污染检测等；

■ 生物医学：X 射线、超声、显微镜图像分析，内窥镜图、温谱图分析，CT 及核磁共振图分析等；

■ 工业生产：无损探伤，石油勘探，生产过程自动化(识别零件，装配，质量检查)，工业机器人视觉的应用与研究等；

■ 气象预报：天气云图测绘、传输；

■ 计算机科学：文字、图像输入的研究，计算机辅助设计，人工智能研究，多媒体计算机与智能计算机研究等；

■ 军事技术：航空及卫星侦察照片的判读，导弹制导，雷达、声纳图像处理，军事仿真等；

■ 侦缉破案：指纹识别，印签、伪钞识别，手迹分析等；

■ 考古：恢复珍贵的文物图片、名画、壁画等的原貌。

# 1.2 图像处理和分析系统

## 1.2.1 系统结构概述

一个基本的图像处理和分析系统可由图 1.1 表示。

从图中可以看出，整个系统可以分为五个模块：图像采集模块、图像处理和分析模块、图像显示模块、图像存储模块和图像通信模块。计算机只能对数字图像进行处理，而自然界能够提供的图像都是模拟类型的，所以首先要通过图像采集模块将图像转化为数字类型，然后再送入处理和分析模块进行处理，处理后的图像不但要经过显示模块形成可视信息，而且还要能够由存储模块保存下来供以后使用。图像的通信模块是随着网络技术的迅猛发展应运而生的。通过图像的传输可以使不同的系统共享图像数据资源，极大地推动了图像在各个领域中的广泛应用。

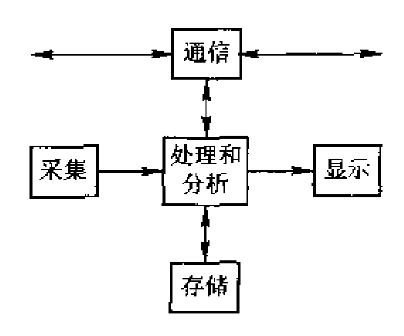


图 1.1 图像处理和分析系统结构示意图

一个好的图像处理与分析系统应该满足以下条件：

■ 系统硬件必须能够适应需要解决的问题，其中最重要的一点就是采集模块必须能够进行正确的空间采样和灰度量化，尽量不引入噪声；

■ 在进行通用图像处理时，处理和分析模块必须能够采用简单而逻辑性强的方法进行处理和分析，以尽量提高图像的处理速度和质量；

■ 图像处理算法库必须具备丰富性、可扩展性，以适应系统处理能力不断增强的要求；

■ 系统要具有足够的存储空间或良好的空间访问技术，这样既能保证在图像处理过程中对临时缓冲的要求，又能够确保对图像的完整存储；

■ 对于图像通信，尤其是图像远程通信来说，一定要保证图像数据量和传输通道相协调，以便获得最快的传输速率。

### 1.2.2 硬件组成

一个典型的计算机图像处理与分析系统主要由以下几个硬件部分组成：图像输入设备、图像输出设备、计算机和显示器。数字图像质量的高低，主要取决于图像输入、输出设备的状况。输入设备性能的高低，如数码相机的镜头质量、分辨率、色位数、存储媒体大小等是影响图像信息源质量的最根本因素；输出设备(如显示器、打印机等)性能的高低则直接决定图像输出质量的好坏。如果需要"看图"，即图像还原，则主要依靠显示器和显卡的作用。图像的各种处理操作，例如，图像滤波、增强和压缩等工作都是由计算机来完成的，因而计算机的性能将直接影响到图像处理的速度和质量。由于图像的数字化过程是图像处理非常基本而重要的一个环节，因而在这里我们着重向大家介绍一些有关图像输入设备的知识，读者可以参考有关书籍来了解其他硬件部分的原理和结构。

图像输入设备也称为图像数字化器，比较常用的有数字摄像机和扫描仪等。一般的图像数字化器由以下几个部分组成：

■ 采样孔：保证能够单独观测特定的像素而不受其他部分的影响；

■ 图像扫描机构：使采样孔能够按照预先指定的方式在图像上移动；

■ 光传感器：能够通过采样孔测量图像的每一个像素的亮度；

■ 量化器：将传感器输出的连续量转化为数字量；

■ 输出存储体：可以是固态存储器或磁盘等。

CCD(电荷耦合器件)摄像机是一种常用的数字化器。对于来自被测对象的自然光通过光学系统，由CCD器件转换成电信号(时间序列的输出信号)，然后将传感器的电荷逐步移出，形成像素。CCD摄像机会引入两种类型的噪声：一种是读出噪声，是由CCD片内电路随机产生的，电荷读出速度越快(曝光时间越短)，亮度越低，其噪声越明显；另一种噪声是光子噪声，是由光的量子性造成的。每个像素每秒接收光子的数目实际上是一个随机数，这个数目一般呈泊松分布，在强曝光或暗电流较大的条件下，光子噪声是图像的主要噪声来源。

扫描仪也是一种图像输入装置，通过扫描胶片上的摄影图像获得数字图像。由于受到胶片上的乳胶颗粒大小和密度的影响，扫描仪存在着固定的额外噪声。通常情况下要求乳胶颗粒较小，乳胶层较薄，以尽量避免乳胶层中的光扩散。

判断一个图像数字化器性能的好坏主要采用以下标准：

■ 像素大小：采样孔的大小和相邻两像素间的间距是数字化器的两个重要性能指标；

■ 图像大小：对于扫描仪器而言，图像大小与允许的输入胶片大小有关，而对于数字化器的输出而言，图像大小是数字化器最大行数与每行最大像素个数的乘积；

■ 线性度：除了要了解灰度正比于图像亮度的精确程度以外，图像可以量化为多少个灰度级也是一个很重要的信息；

■ 噪声情况：图像的噪声越低，数字化器的质量就越好。

### 1.2.3 图像数字化及存储

图像数字化是电脑进行图像处理之前必经的基本步骤，目的是把真实的图像转变成电脑能够接受的处理格式，即特定的一串连续数字。一幅图像只有在空间和灰度上都被离散化后才能够被计算机处理。空间坐标的离散化叫作空间采样，灰度的离散化叫作灰度量化，由此可见，数字化过程是由采样与量化两个步骤组成的。

采样是将时间和空间上连续的图像转换成离散的采样点(即像素)集的过程。事实上，采样就是要决定用多少个点来描述一张图像，采样的结果就是通常所说的图像分辨率。采样可以分为均匀采样和非均匀采样，比较常用的是均匀采样，但是很多情况下可以根据图像特性利用自适应的采样过程来改进图像的视觉效果。例如，在图像灰度过渡区比较尖锐的情况下可以采用较密的采样，而较平滑的区域就可以使用稀疏的采样。采样时要注意采样间隔的选取，采样间隔越小，图像越精细，图像的点越多，其采样数据越大，因而对计算机的负担也就越重。例如，一幅320×240的图像，就表示这幅图像是由76 800个像素点所组成的。可见，如果想要得到更加清晰的图像效果(也就是使这幅图像具有较高的分辨率)，就需要使用更多的点来表示图像，那么相应地就需要付出更大的存储空间。采样通常使用采样频率来进行标示。采样频率是指一秒钟内采样的次数，它反映了采样点之间的间隔大小。丢失的信息越少，采样频率越高，采出的样本就越细腻、逼真，图像的质量越高，但要求的存储量也越大。

将像素点上的灰度值离散为整数，称之为量化，量化的结果是图像容纳的所有颜色数据。量化决定使用多大范围的数值来表示图像采样之后的每一个点，这个数值范围确定了图像能使用的颜色总数。例如，以4个bit存储一个点，就表示图像只能有16种颜色。数值范围越大，表示图像可以拥有更多的颜色，自然就可以产生更为细致的图像效果，但是相应地也必须占用更大的存储空间。量化又分为均匀量化和非均匀量化。均匀量化是指简单地在灰度范围内等间隔量化；非均匀量化是对像素出现频度少的部分采用大的间隔，而频度大的部分采用小间隔。通常所说的量化等级，是指每幅图像样本量化后一共可取多少个像素点(离散的数值)或用多少个二进制数位来表示，它反映了量化的质量，若每个样本用8位(通道数)二进制数表示，则有$2^8$(即256)个量级；若采用16位(通道数)二进制数表示，则有$2^{16}$(即65 536)个量级；若采用24位(通道数)二进制数表示，则有$2^{24}$(即1667万)个量级，同样，量级越大，图像质量就越高，存储空间要求就越大。

计算机图像数字化的质量采用三个方面的主要参数来进行衡量：采样频率、图像样本量化等级及通道数。由于计算机的工作速度、存储空间是相对有限的，各种参数都不能无限提高。那么数字化后的图像数据是如何存储的呢？计算机一般采用两种存储方式：一种是位映射(Bitmap)，即位图存储模式；另一种是向量处理(Vector)，也称矢量存储模式。

位映射是将图像的每一个像素点转换为一个数据，并存放在以字节为单位的矩阵中。当图像是单色时，一个字节可存放8个像素点的图像数据；16色图像每两个像素点用一个字节存储；256色图像每一个像素点用一个字节存储，这样就能够精确地描述各种不同颜色模式的图像画面。所以此种存储模式较适合于内容复杂的图像和真实的照片，例如，用数码相机和扫描仪获取的图像一般都存储为位图。位图图像的缺点在于：随着分辨率以及颜色数的提高，位图图像所占用的磁盘空间会急剧增大，同时在放大图像的过程中，图像

也会变得模糊而失真。

向量处理存储图像内容的轮廓部分，而不存储图像数据的每一点。例如，对于一个圆形图案，只要存储圆心的坐标位置和半径长度，以及圆形边线和内部的颜色即可。该存储方式的缺点是经常耗费大量的时间做一些复杂的分析演算工作，但图像的缩放不会影响显示精度，即图像不会失真，而且图像的存储空间较位图方式要少得多。所以，向量处理比较适合存储各种图表和工程设计图。

总体来看，位图是记录每一个像素的颜色值，再把这些像素点组合成一幅图像，而矢量图是保存图形对象的位置和曲线、颜色的算法。位图占用的存储空间较矢量图要大得多，而矢量图的显示速度较位图慢。

### 1.2.4　图像处理软件

能够进行图像处理的软件很多，其中最著名的就是 Photoshop，该软件能够实现图像显示、增强、剪切、滤波等一系列的操作，效果非常不错，由该软件生成的图像文件格式目前已经作为国际标准予以应用。但是，许多人将图像处理与 Photoshop 画上了等号，这是一种非常错误的概念。事实上，Photoshop 只是一种通用的图像处理软件，一般是用来提高视觉感受，而实际的图像处理工作主要是针对不同的应用领域提取不同的信息的，这是 Photoshop 不善长甚至不支持的。针对不同的应用领域需要不同的图像处理算法，因此用户最好还是根据对某个图像处理软件或高级程序语言的掌握程度来编写自己的图像处理程序。本书将向大家主要介绍如何使用 MATLAB 6.x 来实现各种类型的图像处理。

## 1.3　MATLAB 图像处理初步

MATLAB 是一种基于向量(数组)而不是标量的高级程序语言，因而 MATLAB 从本质上就提供了对图像的支持。从图像的数字化过程我们知道，数字图像实际上就是一组有序的离散数据，使用 MATLAB 可以对这些离散数据形成的矩阵进行一次性的处理。较其他标量语言而言，这是非常有优势的一点。为了使读者对 MATLAB 图像产生一个整体概念，下面我们给出两个图像处理的实例，例子中涉及到的图像处理技术读者可以在后续章节进行学习和掌握。

**⚠ 注意：**

*在启动 MATLAB 运行本书的所有例子之前，必须保证图像处理工具箱已经安装。*

**例 1.1**：图像处理的基本操作。

**步骤一**：读入并显示一幅图像。首先清除 MATLAB 所有的工作平台变量，关闭已打开的图形窗口：

```
clear;
close all;
```

然后使用图像读取函数 imread 来读取一幅图像。假设要读取图像 pout.tif(该图像是图像处理工具箱自带的图像)，并将它存储在一个名为 I 的数组中：

```
I=imread('pout.tif');
```

使用 imshow 命令来显示数组 I：

```
imshow(I)
```

显示结果如图 1.2 所示。



图 1.2　图像 pout.tif 的显示效果

**步骤二**：检查内存中的图像。使用 whos 命令来查看图像数据 I 是如何存储在内存中的：

```
whos
```

MATLAB 将做出如下的响应：

```
Name     Size       Bytes    Class
I        291×240    69840    uint8 array
Grand total is 69840 elements using 69840 bytes
```

**步骤三**：实现直方图均衡化。如图 1.2 所示，pout.tif 是一幅对比度较低的图像。为了观察图像当前状态下亮度的分布情况，用户可以使用 imhist 函数创建描述该图像灰度分布的直方图。使用 figure 命令创建一个新的图像窗口，从而避免直方图覆盖图像数组 I 的显示结果：

```
figure, imhist(I)
```

绘制结果如图 1.3 所示。

从图 1.3 中可以看出，由于图像的灰度范围是比较狭窄的，没有覆盖整个灰度范围[0，255](图像的默认存储类型为 unit8)，并且图像中灰度值的高低区分不明显，因而不能产生好的对比度。下面调用 histeq 函数将图像的灰度值扩展到整个灰度范围中，从而使数组 I 的对比度提高。修改过的图像数据将保存在变量 I2 中：

```
I2 = histeq(I);
```

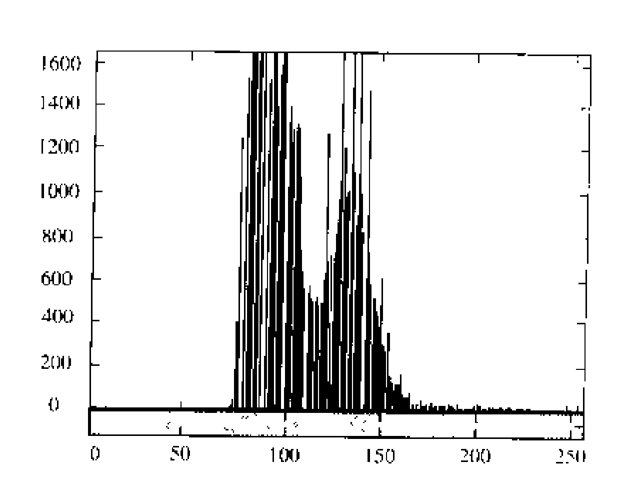



图 1.3　图像 pout.tif 的灰度直方图

在一个新的图像窗口中显示新的经过拓展处理的图像 I2：

```
figure, imshow(I2)
```

拓展后的图像显示结果如图 1.4 所示。

**步骤四**：对 I2 再次调用 imhist 函数以观察拓展后的灰度值分布情况。

```
figure, imhist(I2)
```

灰度直方图如图 1.5 所示。

用户通过使用函数 histeq 来调节图像的像素分布，使之能够分布在与图像类型有关的整个取值范围内。对于一幅图像 X，如果存储类型是 uint8，那么相应的取值范围就是[0，255]；如果是 uint16，则取值范围是[0，65535]；如果是双精度，则取值范围是[0，1]。比较图 1.3 和图 1.5 可以看出，I2 的直方图比 I 的直方图要长且平坦，这种铺展直方图的过程就叫作直方图均衡化。



图 1.4　图像 pout. tif 直方图拓展后的效果

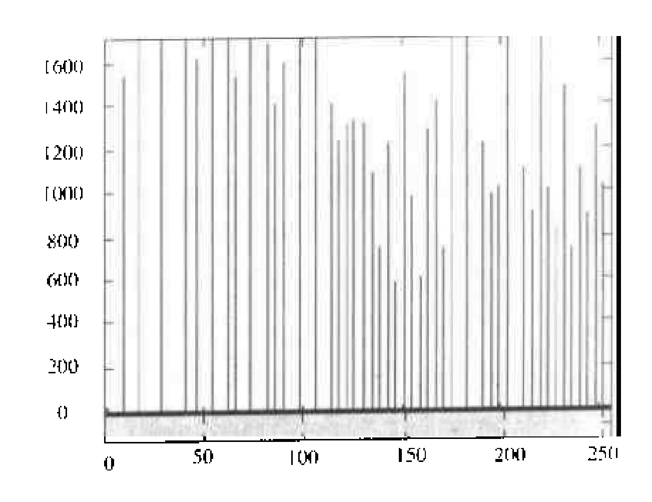



图 1.5　均衡化后的灰度直方图

**步骤五**：保存图像。下面将新的调节后的图像 I2 保存到磁盘中。假设希望将该图像保存为 PNG 格式图像文件，使用 imwrite 函数并指定一个文件名，该文件的扩展名为 .png：

```
imwrite (I2, 'pout2.png');
```

**步骤六**：检查新生成文件的内容。现在使用 imfinfo 函数来观察上述语句写了什么内容到磁盘上。注意，不要在 imfinfo 函数语句行末尾加上分号，以保证 MATLAB 能够显示图像输出结果(仅 imfinfo 函数如此，其它显示函数加不加分号并没有影响，以下不再一一说明)；另外，要保证此时的路径与调用 imwrite 时的路径一致。

```
imfinfo('pout2.png')
```

MATLAB 作如下响应：

```
ans =
    ...
    Filename:'pout2.png'
    FileModDate:'03-Jun-1999 15:50:25'
    FileSize:36938
    Format:'png'
    FormatVersion:[]
    Width:240
    Height:291
    BitDepth:8
    ColorType:'grayscale'
```

本例仅仅说明了 imfinfo 所有返回参数域的一个子集。

**例 1.2**：图像处理的高级应用。

在这个练习中，我们将对另外一幅灰度图像 rice. tif 进行一些较为高级的操作。本练习的主要目的是消除 rice. tif 图像中亮度不一致的背景，并使用阈值将修改后的图像转换为二值图像，使用成员标记返回图像中对象的个数以及统计特性。

**步骤一**：读取和显示图像。清除 MATLAB 工作平台的所有变量，关闭已打开的图形窗口，读取和显示灰度图像 rice. tif：

```
clear, close all;
I = imread('rice.tif');
imshow(I)
```

显示结果如图 1.6 所示。

**步骤二**：估计图像背景。图像 rice.tif 中心位置的背景亮度要高于其他部分的亮度。使用 imopen 函数和一个半径为 15 的圆盘形结构元素对输入的图像 I 进行形态打开操作。形态打开操作将会删除那些不完全包括在半径为 15 的圆盘中的对象，从而实现背景亮度的估计：

```
background = imopen(I, strel('disk', 15));
```

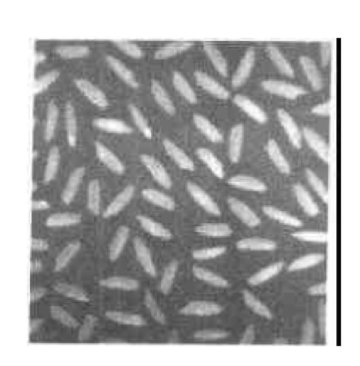
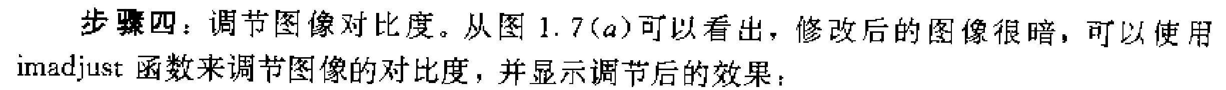


图 1.6　灰度图像 rice.tif 的显示效果

**步骤三**：从原始图像中减去背景图像。现在将背景图像 background 从原始图像 I 中减去，从而创建一个新的、背景较为一致的图像：

```
I2 = imsubtract(I, background);
figure, imshow(I2)
```

显示结果如图 1.7(*a*)所示。



**步骤四**：调节图像对比度。从图 1.7(*a*)可以看出，修改后的图像很暗，可以使用 imadjust 函数来调节图像的对比度，并显示调节后的效果：

```
I3 = imadjust(I2, stretchlim(I2), [0 1]);
figure, imshow(I3);
```

显示结果如图 1.7(*b*)所示。

**步骤五**：使用阈值操作将图像转换为二进制图像。通过使用函数 graythresh 和 im2bw 创建一个新的二值图像 bw：

```
level = graythresh(I3);
bw = im2bw(I3, level);
figure, imshow(bw)
```

显示结果如图 1.7(*c*)所示。

**步骤六**：检查图像中的对象个数。为了确定图像中的米粒个数，使用 bwlabel 函数，该函数标示了二值图像 bw 中的所有相关成分，并且返回在图像中找到的对象个数 numobjects：

```
[labeled, numobjects] = bwlabel(bw, 4);% label components
numobjects =
            80
```

图 1.6 中有些米粒是相互连接的，bwlabel 函数将它们视为同一个对象。

**步骤七**：检查标记矩阵。使用 imcrop 命令来选择并显示已标记的对象和部分背景内的像素。选择一个较小的矩形来进行这项操作，以保证显示的像素值不会引起 MATLAB 命令窗口的滚动。以下语句将使用 imcrop 函数进行交互式的操作。当用户的鼠标位于图像范围内时，其形状会变成十字型，通过点击鼠标并进行拖动来选择一个标记区域，选择完成后，imcrop 函数将显示用户指定的标记区域：

```
grain = imcrop(labeled)
```

观察标记矩阵的一个好办法就是将其显示为一个伪彩色的索引图像。在伪彩色图像中，标记矩阵中的每一个对象都将被映射为相关调色板中的不同颜色，使用函数 label2rgb 来实现这一目的。函数 label2rgb 可以指定调色板、背景颜色以及标记矩阵中的对象将如何被映射为调色板中的颜色：

```
RGB_label = label2rgb(labeled, @spring, 'c', 'shuffle');
imshow(RGB_label);
```

显示结果如图 1.7(*d*)所示。

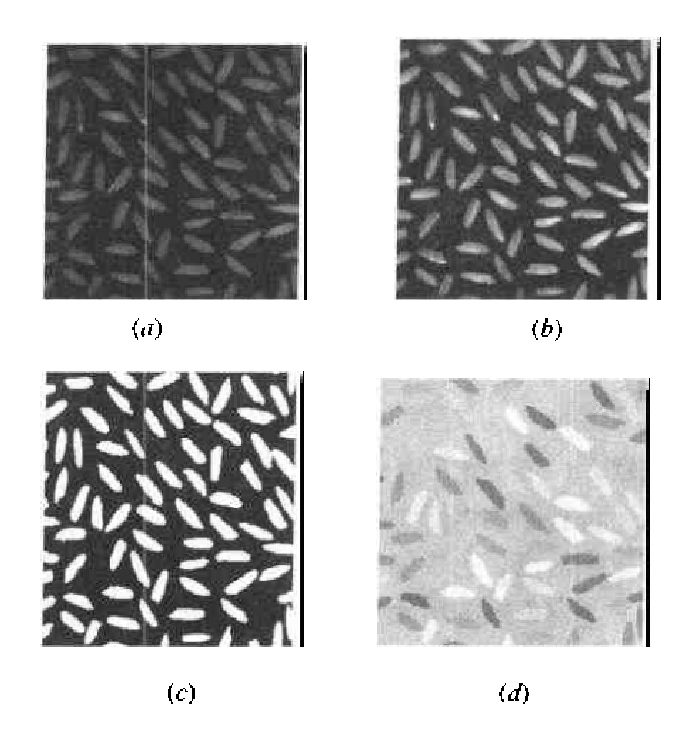


图 1.7　图像处理高级应用的图示效果

(*a*) 去除背景后的效果；(*b*) 调节对比度后的效果；

(*c*) 转换为二进制图像后的效果；(*d*) 伪彩色显示效果

**步骤八**：计算图像中对象的统计属性。

regionprops 命令可以用来调节图像中对象或区域的属性，并将这些属性返回到一个结构体数组中。用户调用 regionprops 函数来返回一个包含图像中所有米粒阈值的基本属性度量结构体。使用以下 MATLAB 函数来计算阈值对象的一些统计属性：首先使用 max 获取最大的米粒大小(如果用户严格按照此处所述的步骤进行操作，那么最终找到的结果将是两个相连的、被标记为同一数值的米粒)：

```
graindata = regionprops(labeled, 'basic')
allgrains = [graindata.area];
max(allgrains)
```

MATLAB 将返回以下数据：

```
ans =
      695
```

使用 find 命令来返回这个最大尺寸米粒的标记号：

```
biggrain = find(allgrains==695)
biggrain =
        68
```

获取米粒的平均大小：

```
mean(allgrains)
ans =
        249
```

绘制一个包含 20 柱的直方图来说明米粒大小的分布情况：

```
hist(allgrains, 20)
```

绘图结果如图 1.8 所示。

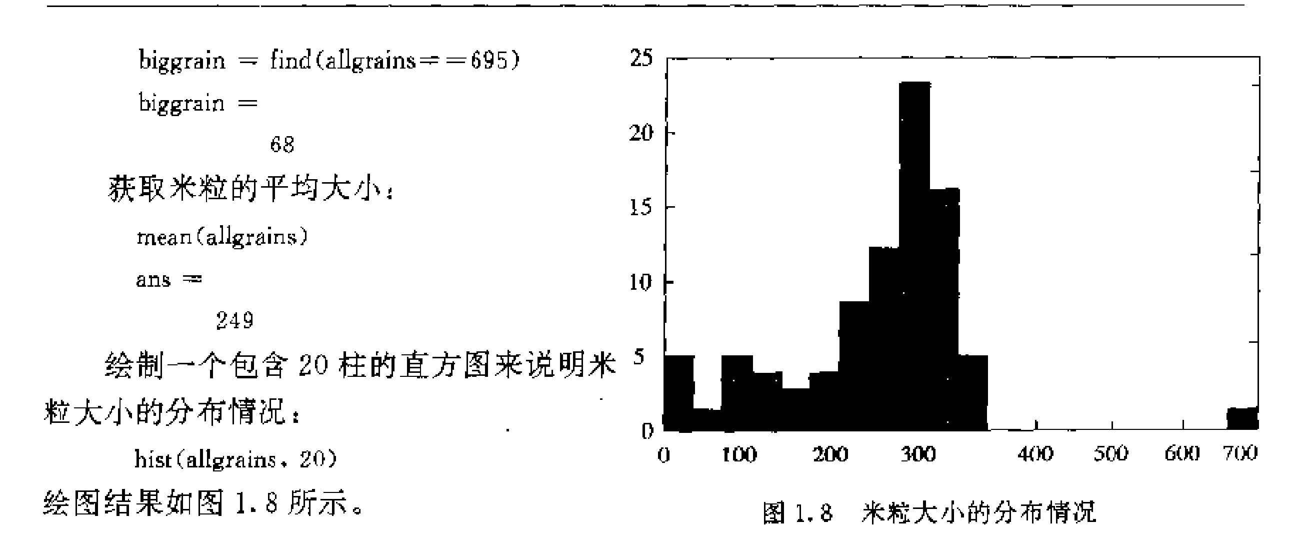



图 1.8 米粒大小的分布情况

# 1.4 图像格式与 MATLAB 图像类型

## 1.4.1 常用图像格式

在介绍图像格式之前，首先要介绍有关图像格式的一个重要概念：调色板。调色板是包含不同颜色的颜色表，每种颜色以红、绿、蓝三种颜色的组合来表示，图像的每一个像素对应一个数字，而该数字对应调色板中的一种颜色，如某像素值为 1，则表示该颜色为调色板的编号为 1 的颜色。调色板的单元个数是与图像的颜色数相对应的，256 色图像的调色板就有 256 个单元。真彩图像的每个像素直接用 R、G、B 三个字节来表示颜色，因此不需要调色板。注意，对于 16 色或 256 色图像并非全部的图像都采用相同的 16 种或 256 种颜色，由于调色板中定义的颜色不同，则不同图像用到的颜色是千差万别的，所谓 16 色或 256 色图像，只是表示该幅图像最多只能有 16 种颜色或 256 种颜色。不同的图像有不同的调色板，在多媒体系统中如果需要同时显示多幅图像时，由于系统在同一时刻只能支持有限的颜色，如使用 256 色驱动程序，则系统以 256 色来同时显示多幅图像，因此，必须使用调色板编辑工具进行调整，以达到不失真地同时显示多幅图像的效果。

图像格式指的是存储图像采用的文件格式。不同的操作系统、不同的图像处理软件，所支持的图像格式都有可能不同。在实际应用中经常会见到以下几种图像格式：

■ BMP(Bitmap)文件：BMP 文件是 Microsoft Windows 所定义的图像文件格式，最早应用在 Microsoft 公司的 Microsoft Windows 窗口系统中。BMP 图像文件有这样一些特点：该结构只能存放一幅图像；只能存储四种图像数据：单色、16 色、256 色、真彩色；图像数据有压缩或不压缩两种处理方式；调色板的数据存储结构较为特殊，其存储格式不是固定的，而是与文件头的某些具体参数(如像素位 bbp、压缩算法等)密切相关的。Windows 设计了两种压缩方式：RLE4 和 RLE8。RLE4 只能处理 16 色图像数据；而 RLE8 则只能压缩 256 色图像数据。BMP 图像文件的文件结构可分为三部分：表头、调色板和图像数据。表头长度固定为 54 个字节。只有真彩色 BMP 图像文件内没有调色板数据，其余不超过 256 种颜色的图像文件都必须设定调色板信息。

■ GIF (Graphics Interchange Format)文件：GIF 文件是由 CompuServe 公司为了方便网络用户传送图像数据而制定的一种图像文件格式。GIF 图像文件经常用于网页的动画、透明等特技制作。GIF 文件有这样一些特点：文件具有多元化结构，能够存储多张图像；调色板数据有通用调色板和局部调色板之分；采用优于 RLE 压缩法的改进版——LZW 压缩法；图像数据一个字节存储一点；文件内的各种图像数据区和补充区多数没有固定的数据长度和存放位置，为了方便程序寻找数据区，就以数据区的第一个字节作为标识符，以使程序能够判断所读到的是哪种数据区；图像数据有两种排列方式：顺序排列和交叉排列；图像最多只能存储 256 色图像。GIF 图像文件结构一般由七个数据单元组成，它们分别是：表头、通用调色板、图像数据区以及四个补充区。表头和图像数据区是文件不可缺少的单元，通用调色板和其余的四个补充区是可选内容。GIF 图像文件可以有多个图像数据区，而每个图像数据区存储一幅图像，通过软件处理和控制使这些分离的图像形成连续有动感的图示效果。

■ TIFF(Tag Image File Format)文件：TIFF 文件是由 Aldus 公司与微软公司共同开发设计的图像文件格式。它有这样一些特点：善于应用指针的功能，可以存储多幅图像；文件内数据区没有固定的排列顺序，但规定表头必须在文件前端，标识信息区和图像数据区在文件中可以随意存放；可制定私人用的标识信息；除了一般图像处理常用的 RGB 模式之外，TIFF 图像文件还能够接受 CMYK、YcbCr 等多种不同的图像模式；可存储多份调色板数据；调色板的数据类型和排列顺序较为特殊；能提供多种不同的压缩数据的方法，以方便使用者选择；图像数据可分割成几个部分进行分别存档。TIFF 图像文件主要由三部分组成：表头、标识信息区和图像数据区。文件内固定只有一个表头，且一定位于文件前端。表头有一个标志参数指出标识信息区在文件中的存储地址，标识信息区有多组标识信息用于存储图像数据区的地址。每组标识信息长度固定为 12 个字节，前 8 个字节分别代表标识信息的代号(两个字节)、数据类型(两个字节)、数据量(四个字节)，最后 4 个字节则存储数据值或标志参数。文件末尾有时还存放一些标识信息区容纳不下的数据，例如，调色板数据就是其中的一项。

■ PCX 文件：PCX 文件是由 Zsoft 公司在 20 世纪 80 年代初期设计的，专用于存储该公司开发的 PC Paintbrush 绘图软件所生成的图像画面数据。目前 PCX 文件已成为 PC 机上较为流行的图像文件。PCX 图像文件具有这样一些特点：一个 PCX 图像文件只能存放一张图像画面；使用 RLE 压缩方法进行数据压缩；PCX 图像文件有多个版本，能处理多种不同模式下的图像数据；4 色和 16 色 PCX 图像文件有可设定或不设定调色板数据的两种选项；16 色图像数据可由一个或四个 bit Plane(颜色的 RGB 等级)来处理。

■ JPEG 格式：它是由 Joint Photographic Experts Group 制定的图像压缩格式，其正式名称为“连续色调静态图像的数字压缩和编码”，是一种通用的静态图像压缩编码标准，可以用不同的压缩比例对文件格式进行压缩。JPEG 压缩技术十分先进，它采用最少的磁盘空间来得到较好的图像质量。

■ PSD 格式：这是 Adobe 公司开发的图像处理软件 Photoshop 中自建的标准图像文件格式，由于 Photoshop 软件被广泛地应用，因而这种格式也很流行。

■ PCD 格式：这是 KODAK 公司所开发的 Photo CD 专用存储格式，由于其文件特别大，所以不得不存在 CD-ROM 上，但其应用领域特别广。

■ WMF 矢量格式：这是微软公司开发的矢量图形格式，在 Office 等软件中得到了大量的应用。

### 1.4.2 MATLAB 图像类型

图像类型是指数组数值与像素颜色之间定义的关系，注意其与图像格式概念的区别。MATLAB 图像处理工具箱支持五种类型的图像，下面我们将介绍。

**说明：**

MATLAB 使用三种存储格式来存储图像：uint8(8 位无符号整数)、uint16(16 位无符号整数)和双精度。在本书中未特别说明的函数都可以对任意一种存储类型进行操作。

#### 1. 二进制图像

在一幅二进制图像中，每一个像素将取两个离散数值(0 或 1)中的一个，从本质上说，这两个数值分别代表状态“开”(on)或“关”(off)。

二进制图像能够使用 uint8 或双精度类型的数组来存储(图像处理工具箱不支持 uint16 类型的二进制图像)。uint8 类型的数组通常比双精度类型的数组性能更好，因为 uint8 数组使用的内存要小得多。在图像处理工具箱中，任何返回一幅二进制图像的函数都使用 uint8 逻辑数组存储该图像。工具箱使用一个逻辑标志来指示 uint8 逻辑数组的数据范围。如果逻辑状态为“开”(on)，那么数据范围是[0，1]；如果为“关”(off)，那么工具箱假定数据范围为[0，255]。图 1.9 给出了一幅典型的二进制图像。

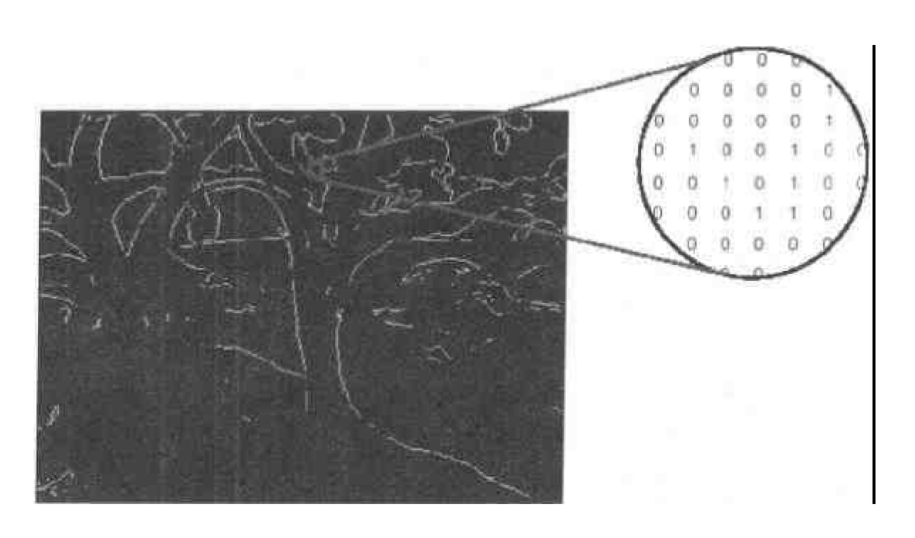


图 1.9　典型二进制图像示例

#### 2. 索引图像

索引图像是一种把像素值直接作为 RGB 调色板下标的图像。

在 MATLAB 中，一幅索引图像包含一个数据矩阵 X 和一个调色板矩阵 map，数据矩阵可以是 uint8、uint16 或双精度类型的，而调色板矩阵则总是一个 m×3 的双精度类型矩阵(其中，m 表示颜色数目)，该矩阵的元素值都是[0，1]范围内的浮点数。map 矩阵的每

一行指定一个颜色的红、绿、蓝颜色分量。索引图像可把像素值直接映射为调色板数值，每一个像素的颜色通过使用 X 的数值作为 map 的下标来获得：数值 1 表示 map 的第一行，数值 2 表示 map 的第二行，以此类推。

调色板通常与索引图像存储在一起，装载图像时，调色板将和图像数据一同自动装载。图 1.10 说明了一幅索引图像的结构，图中的像素由整数表示，这个整数将作为存储在调色板中的颜色数据的指针。

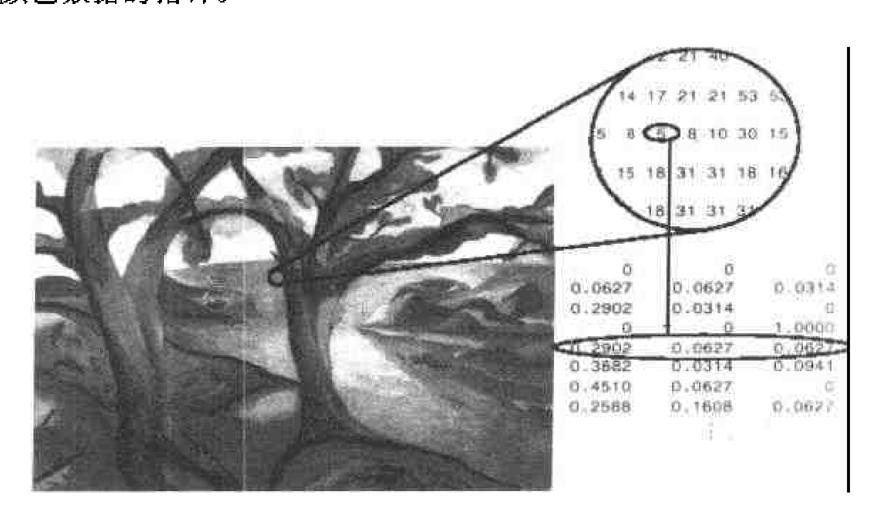



图 1.10　索引图像调色板与数据示例

图像矩阵中的数值与调色板的关系依赖于图像矩阵的类型：如果图像矩阵是双精度类型的，那么数值 1 将指向调色板的第一行，数值 2 将指向调色板的第二行，以此类推；如果图像矩阵是 uint8 或 uint16 类型的，那么将产生一个偏移量：数值 0 表示调色板的第一行，数值 1 表示调色板的第二行，以此类推。在以上的图像中，图像矩阵是双精度类型的，因而没有偏移量，数值 5 就代表调色板的第五行。

**注意：**

图像处理工具箱对 uint16 类型索引图像的支持是有限制的，它可以将 uint16 类型的图像读入 MATLAB 并显示，但是在处理该图像之前必须首先将它转换为 uint8 或双精度类型。如果希望转换为双精度类型，调用 im2double 函数；如果希望将图像转换为 256 色(或更少的颜色)，则调用 imapprox 函数。

#### 3. 灰度图像

灰度图像是包含灰度级(亮度)的图像。在 MATLAB 中，灰度图像由一个 uint8、uint16 或一个双精度类型的数组来描述。

灰度图像实际上是一个数据矩阵 I，该矩阵的每一个元素对应于图像的一个像素点，元素的数值代表一定范围内的灰度级，通常 0 代表黑色，1、255 或 65 535(针对不同存储类型)代表白色。数据矩阵 I 可以是双精度、uint8 或 uint16 类型的。由于灰度图像存储时不使用调色板，因而 MATLAB 将使用一个默认的系统调色板来显示图像。图 1.11 描述了一幅典型的双精度灰度图像。

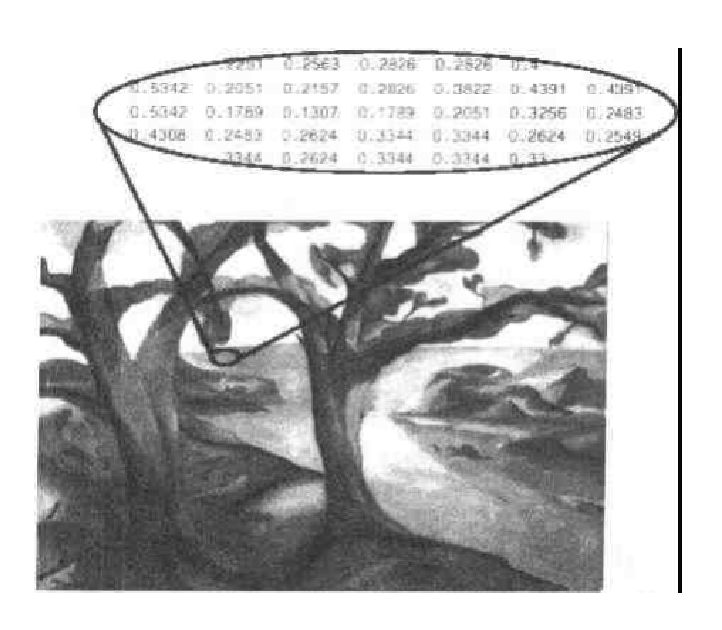


图 1.11　典型的双精度灰度图像示例

**4. 多帧图像**

多帧图像也称为多页图像，是一种包含多幅图像或帧的图像文件。在 MATLAB 内存中，多帧图像是一个四维数组，第四维用来指定帧的序号。多帧图像主要用于需要对时间或场景上相关图像集合进行操作的场合，例如，磁谐振图像切片或电影帧等。

MATLAB 图像处理工具箱提供在同一个数组中存储多幅图像的支持，每一幅单独的图像称为一帧。如果一个数组包含多帧，那么这些图像在第四维中是相联系的。例如，一个包括五幅 400×300 大小的 RGB 图像是一个 400×300×3×5 的数组，而相同帧数的灰度或索引图像将是一个 400×300×1×5 的数组。

利用 cat 命令可以将不同的图像存储到同一个多帧文件中。cat 命令可以将第二、三…个参数按照其第一个参数指定的维数连接起来。例如，cat(2，A1，A2)等价于[A，B]，而 cat(1，A1，A2)等价于[A；B]。又如，如果用户有一组图像 A1、A2、A3、A4 和 A5，可以使用以下命令将这些图像存储为一个简单的数组：

```
A = cat(4, A1, A2, A3, A4, A5)
```

用户还可以从一幅多帧图像中抽取任意一帧图像。例如，以下命令将从多帧图像 MULT1 中抽取第三帧：

```
FRM3 = MULT1(:, :, :, 3)
```

在一个多帧图像数组中，每一幅图像必须有相同的大小和颜色分量。在多帧索引图像中，每一幅图像还要使用相同的调色板。另外，图像处理工具箱中的许多函数(例如，imshow)只能够对多帧图像矩阵的前两维或前三维进行操作，用户也可以对四维数组使用这些函数，但是必须单独处理每一帧。如果用户将一个数组传递给一个函数，并且数组的维数超过该函数的设计操作维数，那么得到的结果是不可预知的：有些情况下，函数将简单地处理数组的第一帧，其他情况下，函数操作将不会产生任何有意义的结果。

**5. RGB 图像**

RGB 图像也称为真彩图像，其每一个像素由三个数值来指定红、绿和蓝颜色分量。

在 MATLAB 中，一幅 RGB 图像由一个 uint8、uint16 或双精度类型的 m×n×3 数组(通常称为 RGB 数组)来描述，其中，m 和 n 分别表示图像的宽度和高度。RGB 图像不使用调色板。每一个像素的颜色由存储在相应位置的红、绿、蓝颜色分量共同决定。RGB 图像是 24 位图像，红、绿、蓝分量分别占用 8 位，因而图像理论上可以包含 $2^{24}$种不同的颜色，由于这种颜色精度能够再现图像的真实色彩，所以称 RGB 图像为真彩图像。

在一个双精度类型的 RGB 数组中，每一个颜色分量都是一个[0，1]范围内的数值，颜色分量为(0，0，0)的像素将显示为黑色，颜色分量为(1，1，1)的像素将显示为白色。每一个像素的三个颜色分量都存储在数据数组的第三维中。例如，像素(10，5)的红、绿、蓝颜色分量分别存储在 RGB(10，5，1)、RGB(10，5，2)和 RGB(10，5，3)中。

图 1.12 给出了一幅典型的双精度 RGB 图像。为了确定位于(2，3)的像素的颜色，需要察看一组数据 RGB(2，3，1:3)("："操作符可以用来直接引用地址从 1～3 的向量数据)。假设(2，3，1)数值为 0.5176，(2，3，2)数值为 0.1608，(2，3，3)数值为 0.0627，那么像素(2，3)的颜色为(0.5176，0.1608，0.0627)。

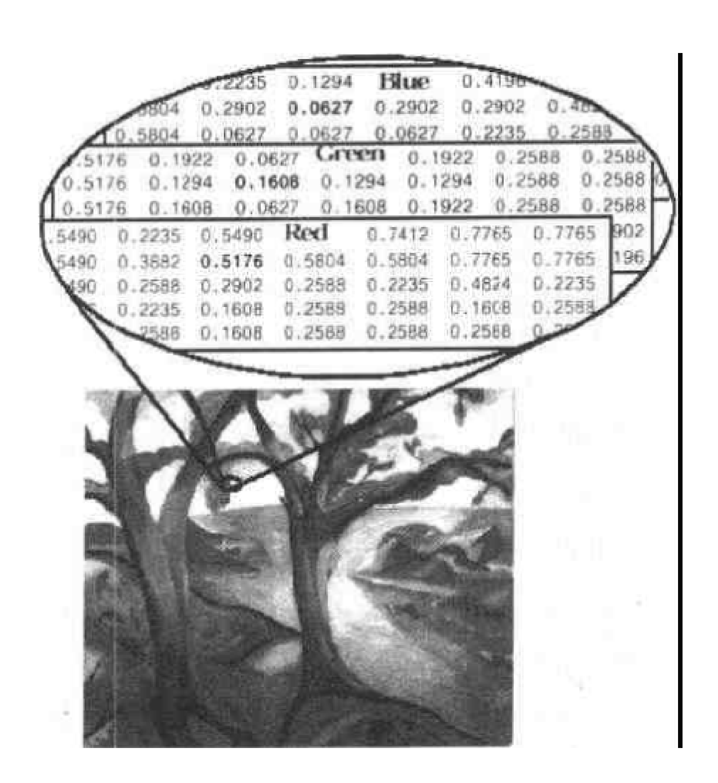



图 1.12 典型的双精度 RGB 图像示例

为了更好地说明在 RGB 图像中所使用的三个不同颜色分量的作用效果，下面我们来创建一个简单的 RGB 图像，该图像包含某一范围内不中断的红、绿、蓝颜色分量，另外，针对每一个颜色分量各创建一幅图像来加以对比：

```
RGB=reshape(ones(64,1) * reshape(jet(64), 1, 192), [64, 64, 3]);
R=RGB(:, :, 1);
```

```
G=RGB(:, :, 2);
B=RGB(:, :, 3);
imshow(R)
figure, imshow(G)
figure, imshow(B)
figure, imshow(RGB)
```

四幅图像的显示结果如图 1.13 所示。

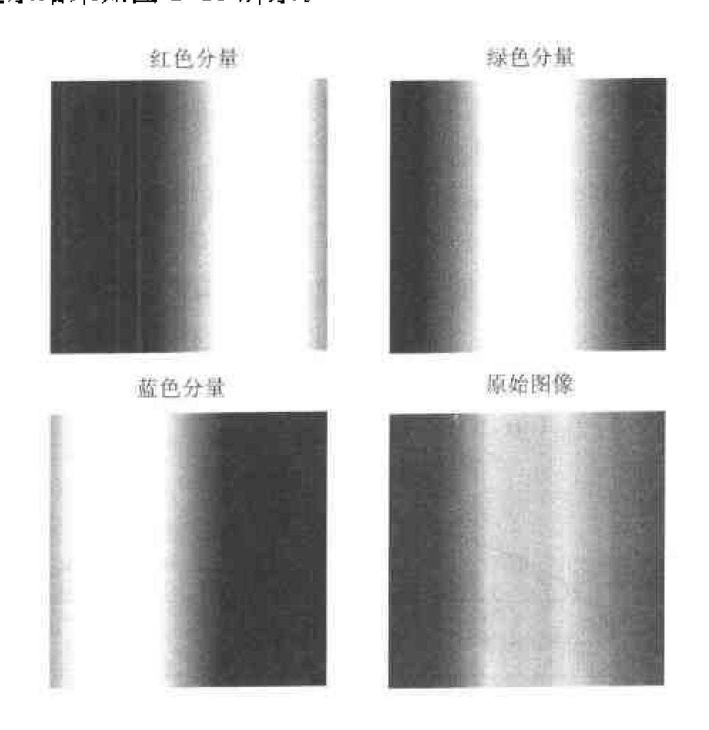



图 1.13　RGB 图像与其颜色分量的显示效果

注意，窗口中每一个单独的颜色项对应的图像都包含一段白色区域，这段白色相应于每一个颜色项的最高值。例如，在红色分量图像中，白色代表纯红色数值浓度最高的区域。当红色与绿色或蓝色混合时将会出现灰像素。图像中的黑色区域说明该区域不包含任何红色数值，即 R=0。

### 1.4.3　MATLAB 图像类型转换

对于某些操作来说，将一幅图像转换为另一种图像类型有时非常有用。例如，如果用户希望对一幅存储为索引图像的彩色图像进行滤波，那么应该首先将该图像转换为 RGB 格式，此时再对 RGB 图像使用滤波器时，MATLAB 将恰当地滤掉图像中的部分灰度值。如果用户企图对一幅索引图像进行滤波，那么 MATLAB 只能简单地对索引图像矩阵的下标进行滤波，这样得到的结果将是毫无意义的。

**注意：**

当用户将一幅图像从一种类型转换为另一种类型时，得到的图像结果可能与原图像效果会有所不同。例如，如果用户将一幅彩色索引图像转换为灰度图像，得到的结果将是一幅灰度级图像，而并不是一幅彩色图像。

表 1.1 给出了图像处理工具箱中所有的图像类型转换函数。

**表 1.1　MATLAB 图像类型转换函数及其功能**

| 函　数 | 功　能 |
| --- | --- |
| dither | 使用抖动方法，根据灰度图像创建二进制图像或根据 RGB 图像创建索引图像 |
| gray2ind | 根据一幅灰度图像创建索引图像 |
| grayslice | 使用阈值截取方法，根据一幅灰度图像创建索引图像 |
| im2bw | 使用阈值截取方法，根据一幅灰度图像、索引图像或 RGB 图像创建二进制图像 |
| ind2gray | 根据一幅索引图像创建一幅灰度图像 |
| ind2rgb | 根据一幅索引图像创建一幅 RGB 图像 |
| mat2gray | 通过数据缩放，再根据矩阵数据创建一幅灰度图像 |
| rgb2gray | 根据一幅 RGB 图像创建一幅灰度图像 |
| rgb2ind | 根据一幅 RGB 图像创建一幅索引图像 |

表 1.1 中几乎所有函数都具有类似的调用格式：函数的输入参数是图像数据矩阵(如果是索引图像，那么输入参数还包括调色板)，返回值是转换后的图像(包括索引图像的调色板)，例如，B[IND，COL]=rgb2ind(RGB)。只有函数 im2bw 的调用格式略有不同，其输入参数还包括一个截取阈值，超过这个阈值的像素被截取为 1，否则为 0，例如：

```
BW = im2bw(X, map, 0.4);
```

下面以函数 grayslice 为例说明这些函数的使用方法。以下代码的功能是根据图 1.14(*a*)所示的灰度图像来创建一幅索引图像(显示结果如图 1.14(*b*)所示)：

```
I = imread('ngc4024m.tif');
X = grayslice(I, 16);
imshow(I)
figure, imshow(X, jet(16))
```




(*a*)　　(*b*)

图 1.14　grayslice 根据灰度图像创建索引图像效果对比

(*a*) 原灰度图像；(*b*) 创建的索引图像

还可以使用 MATLAB 的基本语句来实现某些转换操作。例如，在灰度图像矩阵的第三维上连接该矩阵的三个拷贝就可以得到一幅 RGB 图像：

```
RGB = cat(3, I, I, I);
```

除了以上的标准转换方法之外，还可以利用某些函数返回的图像类型与输入的图像类型之间不同这一特点进行类型转换。例如，基于区域的操作函数总是返回二进制图像，用这些函数可以实现索引或灰度图像向二进制图像的转换。

MATLAB 图像处理工具箱还提供了图像存储类型间的转换函数，这些函数包括 im2double、im2uint8 和 im2uint16，这些函数可以自动进行原始数据的重新标度和偏移。这三个函数的调用格式非常简单，输入参数为图像矩阵，输出为转换后的图像。例如，以下命令将一个描述双精度 RGB1 图像的矩阵(数据范围为[0，1])转换为一个 uint8 类型的 RGB2 图像矩阵([0，255]范围内)：

```
RGB2 = im2uint8(RGB1);
```

也可以使用 MATLAB 函数对图像存储类型进行转换。例如，double 函数可以将 uint8 或 uint16 数据转换为双精度数据。存储类型间的转换将改变 MATLAB 及其工具箱理解图像数据的方式。如果用户希望转换后得到的数组能够被正确地理解为图像数据，那么在进行转换时需要重新标度或偏移数据。例如，如果将一个 uint16 类型的灰度图像转换为 uint8 类型的灰度图像，那么函数 im2uint8 将对原始图形的灰度级进行量化，换句话说，所有 0～128 之间的原始图像数值都将变为 uint8 图像数据中的 0，而 129～385 之间的数值都为 1，以此类推。当使用一种位数较少的类型来描述数字图像时，通常有可能丢失用户图像的一些信息。例如，一个 uint16 类型的灰度图像能够存储 65 536 种不同的灰度级，但是一个 uint8 类型的灰度图像却只能存储 256 种不同的灰度级。一般这种信息的丢失不会产生严重的后果，因为 256 个灰度级仍然超过人眼所能分辨的色彩数目。

图像格式间的转换可以间接利用图像读写函数来完成：首先使用 imread 函数按照原有图像格式进行图像读取，然后调用 imwrite 函数对图像进行保存，并指定图像的保存格式。例如，将一幅图像由 BMP 格式转换为 PNG 格式，则可以这样实现：首先使用 imread 读取 BMP 图像，然后调用 imwrite 函数来保存图像并指定为 PNG 格式：

```
bitmap = imread('mybitmap.bmp', 'bmp');
imwrite(bitmap, 'mybitmap.png', 'png');
```

## 【习　　题】

1. 波特率是一种常用的离散数据传输量度，表示每秒钟传输的比特数，也就是每秒钟发送的 0 和 1 的总数。假设每次先传输一个起始比特，然后再传输 8 个比特的图像数据，最后传输一个终止比特。现在有一幅 256×256 的 256 灰度级图像，指定波特率为 9600，那么传输这幅图像需要多长时间？若传输一幅 1024×1024 的 16 777 216 色的图像，需要多长时间？

2. 利用 MATLAB 的图像读写函数将一幅真彩图像压缩为 JPEG 图像。

# 第二章 图像显示

**本章要点：**

★ 图像显示概述

★ 数字图像显示

★ MATLAB 图像显示方法

★ MATLAB 特殊显示技术

## 2.1 图像显示概述

图像的显示过程是将数字图像从一组离散数据还原为一幅可见的图像的过程。严格地说，图像的显示在图像处理，尤其是图像分析过程中并不是必需的，因为图像处理和分析过程都是基于图像数据的运算，以数字数据或决策的形式给出处理或分析的结果，其中间过程不一定要求可视。但是图像的显示是提高图像处理分析性能非常有用的一个手段，通过图像的显示，用户可以监视图像处理分析过程，并与处理分析软件交互地控制处理分析过程。

数字图像的显示必须符合人眼的视觉要求。人眼是人类视觉系统的重要组成部分，由晶状体和视网膜组成，前者相当于光学镜头(但是要灵活得多)，后者相当于胶片。视网膜表面分布着许多光接收细胞，这些细胞负责接收光的能量并形成视觉图案。光接收细胞有两种：锥细胞和柱细胞，前者在亮度较高时活跃，可以分辨光的颜色，但数量较少；后者对低亮度较为敏感，不感受颜色，只提供视野的整体信息，这就是为什么人眼在天色较暗时看到的物体都是黑白剪影的原因。

由于数字图像是以亮度点集合的形式显示的，因而眼睛区分不同亮度的能力在显示图像时非常重要。人的色觉的产生是一个复杂的过程，可以用三种基本特征量来区分颜色：辉度、色调和饱和度。辉度与物体的反射率成正比，色调是与混合光谱中主要光的波长相联系的，而饱和度则与色调的纯度有关。为了正确使用颜色并建立统一的标准，需要建立合理的色调空间模型。常用的色彩空间模型有：RGB(红/绿/蓝)色调模型，CMYK(青/洋红/黄/黑)色彩模型，LAB(也称 CIELAB，目标色调说明标准)色调模型，HSB(色相/饱和度/亮度)色调模型，最常用的是 RGB 色调模型。RGB 色调模型是根据人眼锥体接收光线的方法来构造成一个模型的，非常适合于标准显示器的工作人员，它用三组独立的值来定义色调、饱和度和亮度。由红(Red)、绿(Green)、蓝(Blue)三组颜色光相互叠加可形成众多的丰富色彩，三组颜色中的任意一组颜色均有 256 个等级的属性定义值，因此三组颜色叠

加可生成 256×256×256=16M 种颜色空间模型(也称加法颜色空间模型)，能满足视觉彩色世界。计算机都是采用 RGB 模型来显示图像的，了解这一点对真彩图像的处理很有帮助。本书中介绍的显示模式都是基于 RGB 色调模型的。

根据不同色调模型开发的图像显示设备是多种多样的。显示设备的构造和特性是直接决定图像显示质量的重要因素之一，正如我们要求图像的计算机处理不应降低图像的质量一样，也要求显示设备能够显示出清晰的数字图像来。

# 2.2　数字图像显示

## 2.2.1　图像的显示特性

图像显示最重要的特性是图像的大小、光度分辨率、灰度线性、平坦能力和噪声特性等，这些显示特性将共同决定一个数字图像显示系统的质量及其在特定应用中的适用性等性能指标。本节介绍的知识主要是针对显示设备的硬件而言的。

光度分辨率是指系统在每个像素位置产生正确的亮度或光密度的精度。在这方面尤为引人注意的是系统所能产生的离散灰度级数目，该参数部分地依赖于系统用来控制像素亮度所使用的位数(bit)。显然，制造一个能够接受 8 位数据的显示系统非常简单，但是制造一个能够正确显示 256 个灰度级的显示系统相对而言要困难得多。由于显示系统内部的电子噪声几乎是不可避免的，因而通常系统的有效灰度级数都会少于理论灰度级数。

灰度线性是亮度或密度正比于输入灰度级的程度。任何显示系统都有一条输入灰度级与输出亮度的变换曲线，如果希望进行正确的运算操作，那么这条曲线必须是线性的。人眼可以忽略变换曲线中的轻微非线性，但是严重的非线性将会导致信息的丢失和图像质量的下降。

显示系统再现大块等灰度级区域(即平坦区域)的能力是显示系统质量的另一个重要标准，图像显示的平坦与否主要取决于相邻像素是否能够很好地配合连接。这个能力与显示点的形状、点间距和显示系统的噪声密切相关。下面以圆形点区域的平坦性为例，讨论这些因素对平坦能力的影响。使用高斯函数描述这些圆形显示点来分析间距与点半径的关系。假设显示点亮度服从如下的二维高斯分布：

$$p(x,y) = e^{-(x^2+y^2)} = e^{-r^2} \tag{2.1}$$

其中，$r$ 表示点到亮点中心的径向距离。假设 $R$ 为亮度，它等于最大亮度值一半的点的径向距离，则点分布曲线函数可以写为

$$p(r/R) = e^{-(r/R)^2 \ln 2} \tag{2.2}$$

即

$$p(r/R) = 2^{-(r/R)^2} \tag{2.3}$$

那么单个点的亮度分布如图 2.1 所示。从图 2.1 可以看出，只有当点到亮点中心的距离大约是 $R$ 的两倍时，其亮度才降低到峰值的 1%以下，也就是说，当点与点之间的间距不大时，显示点之间必定会发生亮度重叠。

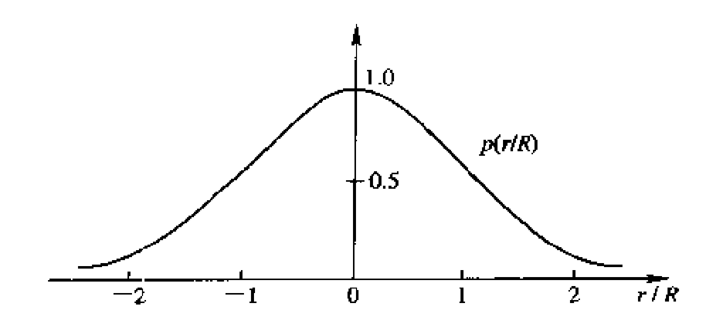



图 2.1　单个点亮度分布

定义某一点处的显示亮度为 $D(x,y)$，该数值表示所有点产生的叠加亮度，即该点的真实亮度。以一个由 12 个像素组成的平坦区域(参见图 2.2(*a*))为例。

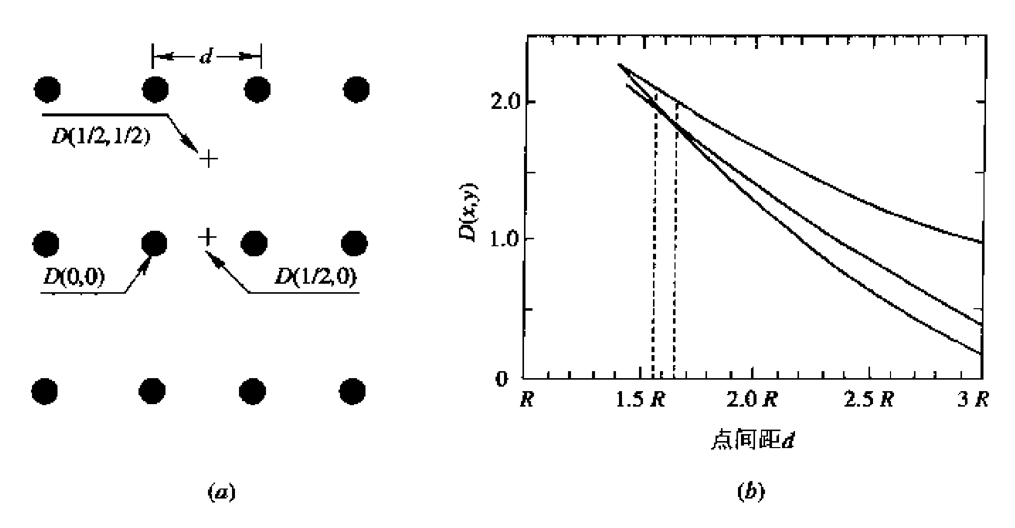



图 2.2　亮点重叠对区域平坦性影响示例

(*a*) 平坦区域；(*b*) 亮点重叠对区域平坦性的影响曲线

通过计算可知

$$D(0,0) = 1 + 4p(d) + 4p(\sqrt{2}d) \tag{2.4}$$

$$D\left(\frac{1}{2},0\right) \approx 2p\left(\frac{d}{2}\right) + 4p\left(\frac{\sqrt{5}d}{2}\right) \tag{2.5}$$

$$D\left(\frac{1}{2},\frac{1}{2}\right) \approx 4p\left(\frac{\sqrt{2}d}{2}\right) + 8p\left(\frac{\sqrt{10}d}{2}\right) \tag{2.6}$$

式中的 $d$ 表示两像素点的间距。

根据以上式子可以绘制出亮点重叠对区域平坦性的影响曲线，如图 2.2(*b*)所示。由图可见，当 $1.55R<d\leqslant1.65R$ 时具有最好的区域平坦性。

显示系统的电子噪声会引起显示点亮度和位置两方面的变化。亮度通道的随机噪声会引起一种“椒盐噪声”的效果，这一现象在平坦区域内尤为明显。来自于显示系统的电子偏转电路的噪声，将会造成点显示间距不平均的现象。这种噪声虽然不会单独产生严重的后果，但是点亮度相互影响效果会放大这种噪声，从而引起相当大的幅值变化，因此必须精确控制像素点的位置。

### 2.2.2　图像的暂时显示

图像的暂时显示是指图像需要复制时，不进行永久拷贝的显示过程。最常见的暂时显示系统是光栅扫描的阴极射线管(CRT)。在 CRT 中，电子枪束的水平位置由计算机控制，每个偏转位置的电子枪束的强度由电压调制，每个点的电压值都与该点的灰度值成正比。另外，如果给一台普通的电视显示器提供合适的视频信号，那么它也可以作为数字图像的暂时显示设备来使用。激光显示器是一种采用移动镜或其他手段偏移光束的设备，可以用 Kerr 元件来调节光的强度，它是一种较为先进的高质量图像显示设备。另外，几种利用液晶和发光二极管技术制造的新型固态显示器正在开发研制当中，将来也是图像暂时显示的可选设备之一。

### 2.2.3　图像的永久显示

图像的永久显示也称为图像记录或图像硬拷贝。采用永久显示技术的 CRT 胶片记录器，实际上是一个装在 CRT 显示器前的照相机。当该照相机的快门打开时，图像像素被依次显示，使得胶片曝光，从而记录图像。另一种常用的显示技术就是打印。打印机有很多种，例如，喷墨打印机、激光打印机、热蜡转移打印机等。

## 2.3　MATLAB 图像显示方法

### 2.3.1　MATLAB 图像的读写和显示

以上介绍了有关图像显示硬件方面的知识，下面介绍如何使用 MATLAB 软件进行正确的图像显示。为此，首先介绍 MATLAB 图像处理工具箱提供的图像读写和显示函数。

函数 imread 可以从任何 MATLAB 支持的图形图像文件格式中以任意位深度读取一幅图像，其基本调用格式如下：

```
[X,MAP] = imread(FILENAME, 'FMT')
```

其中，FILENAME 为需要读入的图像文件名。FMT 为图像格式(可以是 JPG/JPEG、TIF/TIFF、GIF、BMP、PNG、HDF、PCX、XWD、CUR 和 ICO)，如果不指定 FMT 参数，那么系统将根据文件名自动判断图像类型。输出参数 X 表示存储图像数据的矩阵名，当图像为索引图像时，MAP 为该图像的调色板。在实际应用中，我们可以通过使用 imread 函数的在线帮助来获得最新的图像文件格式及其位深度信息。

imread 还可以分帧读取一个多帧图像文件。例如，以下语句将读取 mri.tif 文件的第 5 帧图像：

```
imread ('mri.tif',5);
```

大多数图像文件格式采用 8 位数据存储像素值，将这些文件读入内存后，MATLAB 都将其存储为 uint8 类型。对于支持 16 位数据的文件格式，例如，PNG 和 TIFF，MATLAB 将这些图像存储为 uint16 类型。和其他 MATLAB 生成的图像一样，一旦一幅图像被显示了，那么它将成为一个图形对象句柄。例如，以下代码将图像 ngc6543a.jpg 读入 MATLAB 工作平台，读取数据矩阵为变量 RGB：

```
RGB = imread('ngc6543a.jpg');
```

⚠ 注意：

对于索引图像来说，即使调色板数据本身是 uint8 或 uint16 类型的，imread 函数也要将调色板读入一个双精度类型的数组中。

⚠

函数 imwrite 可以将一幅图像写成一个 MATLAB 支持的格式图形文件。imwrite 函数基本的调用格式如下：

```
imwrite(X,MAP,FILENAME,'FMT')
```

其中，X 为图像变量名，MAP 为调色板，FILENAME 为输出文件名，FMT 为指定的存储格式。如果用户指定文件名时包括了扩展名，那么 MATLAB 将根据这个扩展名推断所需的文件格式。imwrite 函数使用以下的规则来决定输出图像使用的存储种类：如果指定的输出图像文件格式支持 8 位图像，则创建一个 8 位图像文件；如果指定的输出图像文件格式支持 16 位图像，则创建一个 16 位图像文件；如果指定的输出图像文件格式不支持 16 位图像，则用 uint8 来标度图像数据，并创建一个 8 位图像文件(这是因为大多数图像文件格式都使用 8 位)。

例如，以下语句将根据一个 MAP 文件(MATLAB 数据文件)装载一幅小丑图像，然后将其保存为一个包含小丑图像的 BMP 文件。

```
load clown
imwrite(X,map,'clown.bmp')
```

对于某些图像格式，imwrite 函数还提供一些额外的参数来控制图像格式。例如，以下代码将一个灰度图像 I 写为一个位深度为 4 的 PNG 文件：

```
imwrite(I,'clown.png','BitDepth',4);
```

而以下代码将一幅图像 A 保存为一个压缩比设置为 100(缺省值为 75)的 JPEG 文件：

```
imwrite(A, 'myfile.jpg', 'Quality', 100);
```

可以通过调用 imfinfo 函数获得与图像文件有关的信息，其调用格式如下：

```
INFO = imfinfo(FILENAME,'FMT')
```

其中，INFO 是一个结构体，包含与文件的具体类型有关的文件信息，对于每一种类型的文件，INFO 都包含这样一些信息：文件名、文件格式、文件格式版本号、文件的修正数据、文件的字节大小、图像宽度(以像素为单位)、图像高度(以像素为单位)、每个像素所占的位数、图像类型等。

在 MATLAB 中，显示一幅图像的基本方法是使用 image 函数，这个函数将创建一个图形对象句柄，其调用格式如下：

```
image(X,Y,C)
```

其中，X 和 Y 表示图像显示位置的左上角坐标，C 表示需要显示的图像。另一个图像显示函数 imagesc 与 image 函数类似，但是它可以自动标度输入数据。

MATLAB 图像处理工具箱提供了一个高级的图像显示函数 imshow。和 image 与 imagesc 函数一样，这个函数也将创建一个句柄图形图像对象，所不同的是，imshow 函数将自动设置图形对象句柄的属性以及图像特征，从而获得最佳的显示效果。在图像处理应用程序中，使用 imshow 函数比使用 image 和 imagesc 函数要好，因为 imshow 函数的调用

方法比较简单，并且在大多数情况下都是使用一个屏幕像素来显示一个图像像素的。

imshow 函数的基本调用格式如下：

```
imshow(C,MAP,Dis_Opt);
```

其中，C 表示待显示的图像，当 C 为索引图像时才需要 MAP 调色板参数；Dis_Opt 是可选参数，用来设置图像显示时是否显示为真实大小。

这里要注意的是，某些工具箱选项的设置会对 imshow 的显示效果产生影响，这些选项包括：

■ ImshowBorder：图像显示时在图像坐标轴和窗口边界之间是否留有边框；

■ ImshowAxesVisible：是否显示图像的坐标轴及其标记；

■ ImshowTruesize：是否调用 truesize 函数。truesize 函数给每一个图像像素分配一个单独的屏幕像素，也就是说，一个 200×300 的图像将显示为 200 个屏幕像素高，300 个屏幕像素宽；

■ TureSizeWarn：当图像大于屏幕大小时是否发出警告信息。

可以调用 iptsetpref 函数来设置工具箱的这些选项。例如，设置图形窗口大小恰好完全包含图像：

```
iptsetpref('ImshowBorder', 'tight');
```

在大多数情况下，imshow 函数在显示图像之前自动调用 truesize 命令，并将图像显示为真实大小，但是某些情况下，可能不希望调用 truesize 函数，而将图像显示为缺省的坐标轴尺寸。例如，当处理一幅小图像时会希望图像能够放大，在这种情况下，屏幕像素值与图像矩阵的元素之间并不是直接对应的，MATLAB 必须使用插值方法来决定屏幕像素的数值。

有两种方式可以决定 MATLAB 是否自动调用 truesize 命令：一种方法是调用iptsetpref 函数将 MATLAB 工具箱 ImshowTuresize 选项设置为 manual：

```
iptsetpref('ImshowTruesize','manual')
```

另一种方法就是通过设置 imshow 函数的 Dis_Opt 参数值，将其设置为 notruesize，这样在显示图像时将不会调用 truesize 命令：

```
imshow(X, map,'notruesize')
```

**说明：**

*如果希望将本次运行中的设置保存到下次运行中，可以将修改后的工具项的选项设置保存到 startup.m 文件中。*

## 2.3.2 二进制图像的显示方法

使用以下调用格式可以显示二进制图像：

```
imshow(BW)
```

在 MATLAB 中，一幅二进制图像是一个 uint8 或双精度类型的二维逻辑矩阵，该矩阵仅包括数值 0 和 1。工具箱不支持 uint16 类型的二进制图像，所有能够返回二进制图像的工具箱函数都采用 uint8 逻辑数组存储图像数据。通常使用工具箱对二进制图像进行操作是非常简单明了的，在大多数情况下，用户装载一幅 1 位二进制图像，然后 MATLAB 将

在内存中创建一个 uint8 逻辑图像。

正常情况下用户不会遇到双精度类型的二进制图像，除非使用 MATLAB 自己创建双精度图像。如果用户装载的图像每个像素的位深度大于 1，或者自己使用 MATLAB 创建一个新的仅包含 0 和 1 的双精度或 uint8 图像，那么可能会产生估计不到的后果。数据全部都是 0 和 1 并不总是代表数据描述的是一幅二进制图像。如果希望工具箱将图像理解为二进制的，那么所有数据都要是逻辑数据，也就意味着必须对其逻辑标志进行设置（将所有数据标志均设置为 on）。因此，数据碰巧都是 0 和 1 的灰度图像将不会被理解为二进制图像。

下面的例子强调了用户在显示二进制图像时逻辑标志的重要性。创建一个仅包含 0 和 1 的双精度类型的图像：

```
BW1 = zeros(20,20);
BW1(2:2:18,2:2:18)=1;
imshow(BW1, 'notruesize');
```

显示结果如图 2.3 所示。

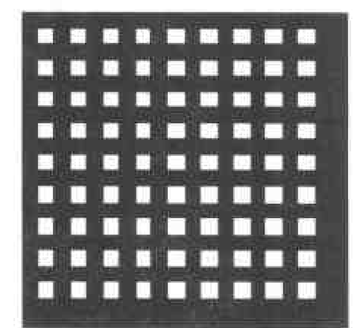


图 2.3 数值全部为 0 或 1 的双精度图像的显示效果

观察图像在内存中的存储格式：

```
whos
    Name     Size      Bytes      Class
    BW1      20×20     3200       double array
Grand total is 400 elements using 3200 bytes
```

虽然图像看起来很像二进制图像，但是从存储格式上可以看出，imshow 函数实际上把该图像视为灰度图像。如果用户将这个图像保存在一个文件中（不指定位深度），那么该图像将被保存为一个 8 位、仅包含 0 和 1 的灰度图像。如果将 BW1 转换为 uint8 类型，那么很明显可以看出 BW1 并不是一幅真正的二进制图像：

```
BW2=uint8(BW1);
figure,imshow(BW2,'notruesize')
```

显示结果将是一幅几乎全黑的图像。虽然 BW1 包含非零元素，但是由于 uint8 灰度图像的颜色动态变化范围在 0～255 之间，因而数值 1 将显示为非常近似于黑色的颜色。

为了使 BW1 成为一个真正的二进制图像，使用不等操作符（～＝）将逻辑标志设置为 on：

```
BW3 = BW2 ~= 0;
figure, imshow(BW3,'notruesize')
```

显示结果将与图 2.3 有相同的效果。

此时再观察图像在内存中的存储格式：

```
whos
    Name     Size      Bytes      Class
    BW1      20×20     3200       double array
    BW2      20×20     400        uint8 array
    BW3      20×20     400        uint8 array (logical)
```

使用某种支持 1 位图像的格式将 BW3 写入一个文件。用户不需要指定位深度，因为如果该格式支持 1 位图像，那么 MATLAB 自动将图像保存为 uint8 或双精度逻辑图像：

```
imwrite(BW3, 'grid.tif');
```

可以通过调用函数 imfinfo 来检查 grid. tif 图像的位深度：

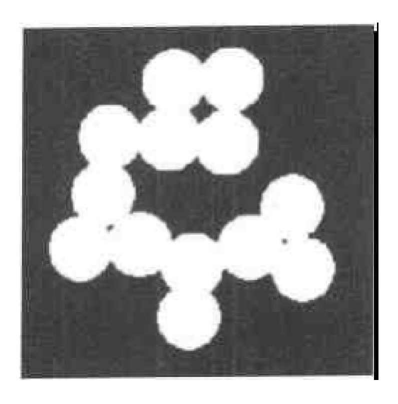
```
ans =
        ...
        Width: 20
        Height: 20
        BitDepth: 1
        ColorType: 'grayscale'
        FormatSignature: [73 73 42 0]
        ...
```

其中，BitDepth 域说明该图像被保存为 1 位图像。读者可能已经注意到，二进制图像的 ColorType 域的查询结果为 grayscale，MATLAB 将这个域设置为三个数值中的一个：grayscale、indexed 和 truecolor。当读取一幅图像时，MATLAB 通过检测 BitDepth 和 ColorType 两个域的数值来判断图像的类型。

在显示二进制图像时，有时可能希望将图像进行逆转，用数值 0 表示白色，1 表示黑色。使用 MATLAB 的"～"操作符(取反)进行这项工作，可以达到预期的效果。例如，对原始二进制图像(图 2.4(*a*))进行取后操作。

```
BW = imread('circles.tif');
imshow(BW);
figure; imshow(~BW);
```

显示结果如图 2.4(*b*)所示。



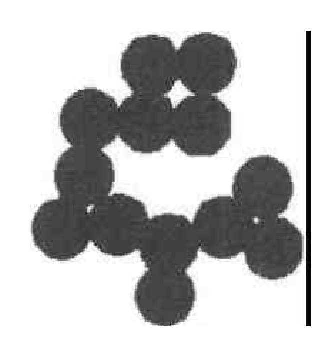


图 2.4　原始二进制图像和取反后的二进制图像的显示效果
(*a*) 原始二进制图像；(*b*) 取反后的二进制图像

用户还可以使用一个调色板来显示一幅二进制图像。如果图形是 uint8 类型的，那么数值 0 将显示为调色板的第一个颜色，数值 1 将显示为第二个颜色。例如，以下命令将数值 0 显示为红色，数值 1 显示为蓝色：

```
imshow(BW,[1 0 0; 0 0 1])
```

显示结果如图 2.5 所示。

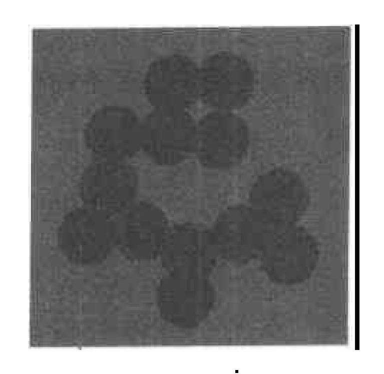


图 2.5　使用调色板的二进制图像的显示效果

如果图像是双精度类型的，那么用户需要给图像矩阵的每一个数值加 1，因为调色板索引此时是没有偏移量的：

```
BW = double(BW);
imshow(BW + 1,[1 0 0; 0 0 1])
```

imshow 通过以下设置来控制图像显示颜色的属性：

- 图像的 CData 属性被设置为 BW 中的数值；
- 图像的 CDataMapping 属性设置为 direct；
- 坐标轴的 CLim 属性设置为[0 1]；
- 图形窗口的 Colormap 属性设置为一个数值范围从黑到白的灰度级调色板。

**说明：**

*以上的图像属性设置是由 imshow 函数自动完成的，用户无需掌握这些内容，以上的说明仅供了解，有兴趣的读者可以参考有关图像对象属性的书籍。*

### 2.3.3 灰度图像的显示方法

显示灰度图像最基本的调用格式如下：

```
imshow(I)
```

imshow 函数通过将灰度值标度为灰度级调色板的索引来显示图像。如果 I 是双精度类型的，那么像素值 0.0 将显示为黑色，1.0 将显示为白色，这两个数值之间的像素值将显示为灰影。如果 I 是 uint16 类型的，那么像素值 65535 将显示为白色。

灰度图像与索引图像在使用 m×3 大小的 RGB 调色板方面是类似的，但是正常情况下无需指定灰度图像的调色板。MATLAB 使用一个灰度级系统调色板(R=G=B)来显示灰度图像。缺省情况下，在 24 位颜色系统中调色板包含 256 个灰度级，在其他系统中包含 64 或 32 个灰度级。

imshow 函数显示灰度图像的另一种调用格式是：可以使用户明确地指定所使用的灰度级数目。例如，以下语句将显示一幅 32 个灰度级的图像 I：

```
imshow(I,32)
```

由于 MATLAB 将自动对灰度图像进行标度以适合调色板的范围，因而对灰度图像可以使用自定义大小的调色板。在某些情况下，可能需要将一些超出数据惯例范围(对于双精度数组为[0,1]，对于 uint8 数组为[0,255]，对于 uint16 数组为[0,65535])的数据显示为一幅灰度图像。例如，如果用户对一幅灰度图像进行滤波，那么输出数据的部分值将超过原始图像的数据范围，为了将这些超过惯例范围的数据显示为图像，用户可以直接指定数据的范围，其调用格式如下：

```
imshow(I,[low high])
```

其中，low 和 high 参数分别为数据数组的最小值和最大值。如果用户使用一个空矩阵([ ])指定数据范围，那么 imshow 将自动进行数据标度。以下的例子将对一幅灰度图像进行滤波，从而得到超出惯例范围的数据，然后使用空矩阵调用 imshow 来显示所得的数据：

```
I = imread('testpat1.tif');
J = filter2([1 2;-1 -2],I);
imshow(J,[]);
```

显示结果如图 2.6 所示。

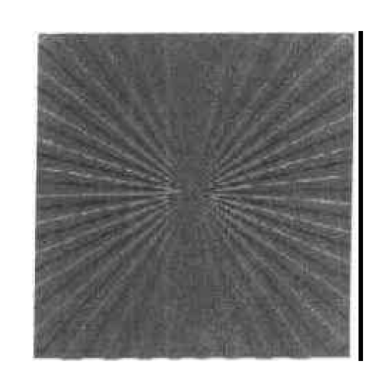


图 2.6 超出惯例范围的数据以灰度图像的形式进行显示的效果

使用这种调用格式，imshow 将坐标轴的 CLim 属性设置为[min(J(:)) max(J(:))]。CDataMapping 对于灰度图像来说总是取值 scaled，因此数值 min(J(:))将使用调色板的第一个颜色来显示，而数值 max(J(:))将使用调色板的最后一个颜色来显示。

imshow 函数将设置以下句柄图形属性来控制颜色显示方式：

■ 图像的 CData 属性被设置为 I 中的数据；

■ 图像的 CDataMapping 属性被设置为 scaled；

■ 如果图像矩阵是双精度类型的，那么坐标轴的 CLim 属性被设置为[0, 1]；如果是 uint8 类型的，那么坐标轴的 CLim 属性被设置为[0, 255]；如果是 uint16 类型的，那么坐标轴的 CLim 属性被设置为[0, 65535]；

■ 图形窗口的 Colormap 属性被设置为数值范围从黑到白的灰度级调色板。

### 2.3.4 索引图像的显示方法

使用 imshow 显示索引图像需要指定图像矩阵和调色板：

```
imshow(X,map)
```

对于 X 的每一个像素，imshow 显示存储在 map 相应行中的颜色。图像矩阵中数值和调色板之间的关系依赖于图像矩阵是双精度、uint8 还是 uint16 类型。如果图像矩阵是双精度类型的，那么数值 1 将指向调色板的第一行，数值 2 指向第二行，以此类推。如果图像矩阵是 uint8 或 uint16 类型的，那么会有一个偏移量：数值 0 指向调色板的第一行，数值 1 指向第二行，以此类推。偏移量是由图像对象自动掌握的，不能通过使用句柄图形属性来进行控制。

索引图像的每一个像素都直接映射为其调色板的一个入口。如果调色板包含的颜色数目多于图像颜色数目，那么调色板中额外的颜色都将被简单地忽略掉；如果调色板包含的颜色数目少于图像颜色数目，那么所有超出调色板颜色范围的图像像素都将被设置为调色板中的最后一个颜色，也就是说，如果有一幅包含 256 色的 uint8 索引图像，而用户使用一个仅有 16 色的调色板来显示这幅图形，那么所有数值大于等于 15 的像素都将被显示为调色板的最后一个颜色。

imshow 函数将设置以下句柄图形属性来控制颜色显示方式：

■ 图像 CData 属性将设置为 X 中的数据；

■ 图像 CDataMapping 属性将设置为 direct(并因此导致坐标轴的 CLim 属性无效)；

■ 图形窗口的 Colormap 属性被设置为 map 中的数据。

### 2.3.5 RGB 图像的显示方法

RGB 图像又称为真彩图像，它直接对颜色进行描述而不使用调色板。显示 RGB 图像最基本的函数格式如下：

```
imshow(RGB)
```

RGB 是一个 m×n×3 的数组。对于 RGB 中的每一个像素(r,c)，imshow 显示数值(r,c,1:3) 所描述的颜色。每个屏幕像素使用 24 位颜色的系统就能够直接显示真彩图像，因为系统给每个像素的红、绿、蓝颜色分量分配 8 位(256 级)。在颜色较少的系统中，MATLAB 将综合使用图像近似和抖动技术来显示图像。

imshow 函数将设置以下句柄图形属性来控制颜色显示方式：

■ 图像的 CData 属性被设置为 RGB 中的数值，这个数值是三维的。当设置 CData 属性是三维时，MATLAB 将数组理解为真彩数据；

■ 忽略图像的 CDataMapping 属性；

■ 忽略坐标轴的 CLim 属性；

■ 忽略图形窗口的 Colormap 属性。

### 2.3.6 磁盘图像的直接显示

通常在显示一幅图像之前，首先要调用 imread 函数来装载图像，将数据存储为 MATLAB 工作平台中的一个或多个变量。如果不希望在显示图像之前装载图像，那么可以使用以下格式直接进行图像文件的显示：

```
imshow filename
```

其中，filename 为要显示的图像文件的文件名。

例如，显示一个名为 flowers.tif 的文件：

```
imshow flowers.tif
```

如果图像是多帧的，那么 imshow 将仅仅显示第一帧，这种调用格式对于图像扫描非常有用。但是要注意，当用户使用这种格式时，图像数据没有保存在 MATLAB 工作平台中，如果用户希望将图像装入工作平台中，使用函数 getimage，该函数将从当前的句柄图形图像对象中获取图像数据。例如：

```
rgb = getimage;
```

如果显示 flowers.tif 文件的图形窗口当前被激活，那么该语句就会将图像赋给变量 rgb。

**注意：**

*要显示的图像文件必须位于当前路径或 MATLAB 的搜索路径中。*

## 2.4 MATLAB 特殊显示技术

### 2.4.1 添加色带

使用 colorbar 函数可以给一个坐标轴对象添加一条色带。如果给一个包含图像对象的坐标轴添加了一条色带，那么色带将对应于图像中使用的不同颜色数值。

如果正在将一些非惯例范围内的数据显示为一幅图像，那么使用色带来观察数据值与颜色之间的相应关系是非常有用的。以下的语句将过滤一幅 uint8 类型的灰度图像，产生

一些超出[0,255]范围的数据，然后将这些数据显示为灰度图像：

```
I = imread('saturn.tif');
h = [1 2 1; 0 0 0; -1 -2 -1];
I2 = filter2(h,I);
imshow(I2,[]), colorbar
```

相应的灰度图像如图 2.7 所示。从图中可以看出数值与颜色的对应关系。

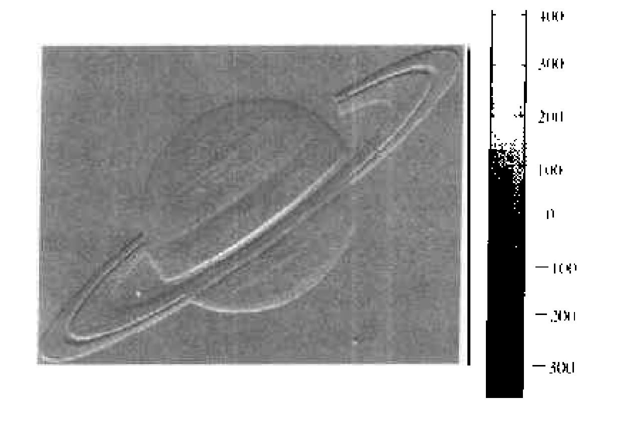


图 2.7　带有色带的非惯例数据灰度图像的显示效果

### 2.4.2　显示多帧图像

多帧图像是一个包含多个图像的图像文件。MATLAB 支持的多帧图像的文件格式包括 HDF 和 TIFF 两种。文件一旦被读入 MATLAB，多帧图像的显示帧数由矩阵的第四维数值来决定。调用 imread 函数的特殊语法格式能够将多帧图像从磁盘装载到 MATLAB 中，也可以使用 MATLAB 函数创建多帧图像，多帧图像能够使用很多种显示方法，其中包括：

- 使用 imshow 函数单独显示每一个图像帧；
- 使用 montage 函数立即显示所有图像帧；
- 使用 immovie 函数将图像帧转换为电影。

在 MATLAB 中，多帧图像的每一帧都是由第四维来控制的。为了观察图像的每一帧画面，可调用 imshow 函数，并使用标准 MATLAB 索引符号来指定画面帧号。例如，如果希望观察灰度数组 I 的第 7 帧，则可调用以下语句：

```
imshow(I(:,:,:,7))
```

以下语句将装载 mri.tif 文件，并显示其第 3 帧画面：

```
mri = uint8(zeros(128,128,1,27));  % 初始化包含 27 帧图像
                                   % 的文件 mri.tif
for frame=1:27
[mri(:,:,:,frame),map] = imread('mri.tif',frame);
                          %将每一帧读入内存中相应的图像帧中
end
imshow(mri(:,:,:,3),map);
```

显示结果如图 2.8 所示。

图 2.8　mri.tif 文件第 3 帧画面的显示效果

灰度、索引和二值多帧图像的维数都是 m×n×1×k，其中 k 表示帧的总数，1 说明该图像数据仅有一个颜色分量。因此，以下的调用格式：

```
imshow(mri(:,:,:,3),map);
```

等价于：

```
imshow(mri(:,:,1,3),map);
```

RGB 多帧图像的维数是 m×n×3×k，3 表示 RGB 图像使用三种颜色分量。例如，以下语句将显示第 7 帧的三个颜色分量：

```
imshow(RGB(:,:,:,7));
```

该语句与以下语句并不等效：

```
imshow(RGB(:,:,3,7));
```

第二条语句仅仅显示第7帧的第三个颜色分量。以上的两种调用格式仅仅在图像为RGB灰度级(R=G=B)时才能产生相同的效果。

如果希望在同一时刻观察图像数据的所有帧，可调用函数montage。montage函数将对图形窗口进行划分，各帧显示在不同的显示区域中。montage的语法格式与imshow函数类似。例如，显示多帧索引图像的格式如下(注意，多帧索引图像中的所有帧都必须使用相同的调色板)：

```
montage(X,map)
```

以下语句将装载一幅多帧索引图像的所有帧并显示：

```
mri = uint8(zeros(128,128,1,27));
for frame=1:27
    [mri(:,:,:,frame),map] = imread('mri.tif',frame);
end
montage(mri,map);
```

显示结果如图2.9所示。

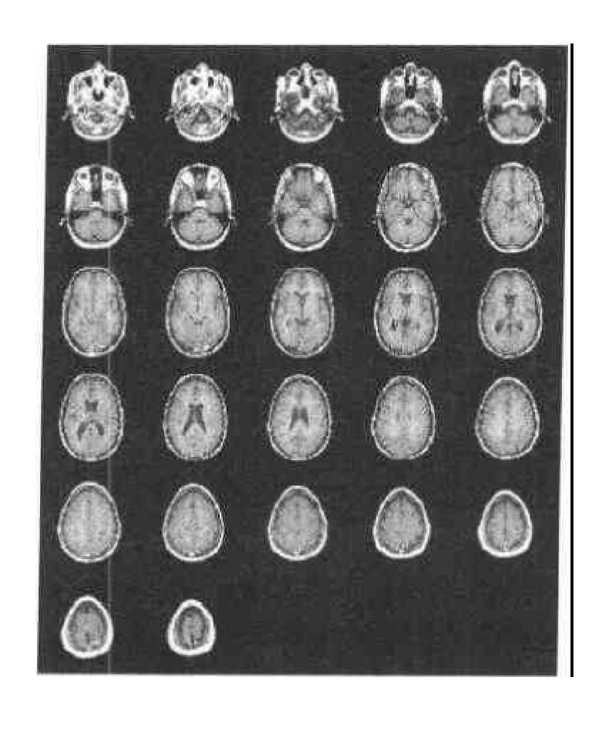


图2.9　mri.tif多帧同时显示的效果

montage函数按行显示图像帧：第一帧位于第一行的第一个位置，第二帧位于第一行的第二个位置，依此类推。montage函数将所有帧近似地排列为正方形。

如果希望根据多帧图像数组创建一个MATLAB电影片断，则可调用函数immovie。以下调用将根据多帧索引图像X创建一个电影片断：

```
mov = immovie(X,map);
```

上式中的 X 是用户电影希望使用的四维图像数组，可以调用 movie 函数在 MATLAB 中显示这段电影：

```
movie(mov);
```

本例将装载多帧图像 mri. tif，并根据图像创建一段电影。读者可以通过使用以下语句体验到非常有趣的显示结果：

```
mri = uint8(zeros(128,128,1,27));
for frame=1:27
    [mri(:,:,:,frame),map] = imread('mri.tif',frame);
end
mov = immovie(mri,map);
movie(mov);
```

immovie 函数在创建电影的同时也会对电影进行显示，所以读者将看到两遍图像显示结果，但第二次的运行速度比第一次要快得多。MATLAB 电影要求图像按顺序显示，为了使一段电影可以在 MATLAB 以外的环境中运行，用户可以使用 MATLAB 的 avefile 函数和 addframe 函数来创建一个 AVI 文件。AVI 文件可以由 uint8 和双精度类型的索引和 RGB 图像来创建。

### 2.4.3　显示多幅图像

MATLAB 没有对用户想要同时显示的图像数目进行限制，然而，由于受到用户的计算机硬件配置的影响，图像的显示数目通常会存在一些系统限制。本节将介绍如何分别显示多个图形窗口或者如何使用同一个窗口显示多幅图像的方法。对图像显示数目的限制主要来源于用户系统能够显示的颜色数目，这主要与系统为每一个像素保存信息所使用的维数有关。许多系统都可以使用 8 位、16 位和 24 位来显示一个像素，如果用户使用 16 位或 24 位显示每一个像素，那么无论显示图像的多少帧都不会遇到什么问题，但是如果使用的是 8 位显示系统，那么最多可以显示 256 种不同的颜色，因此用户在显示多帧图像时会很快耗尽颜色通道(事实上可用的颜色通道少于 256，因为部分颜色通道被保留用于控制句柄图形图像对象，操作系统通常也会保留一些颜色通道)。

为了判断系统颜色的显示位数，可以使用以下命令：

```
get(0,'ScreenDepth')
```

显示多幅图像最简单的方法就是在不同的图形窗口中显示它们。imshow 函数总是在当前窗口中显示一幅图像，如果用户想连续显示两幅图像，那么第二幅图像就会替代第一幅图像。为了避免图像在当前窗口中的覆盖现象，在调用 imshow 函数显示下一幅图像之前可使用 figure 命令来创建一个新的空图形窗口。例如：

```
imshow(I)
figure, imshow(I2)
figure, imshow(I3)
```

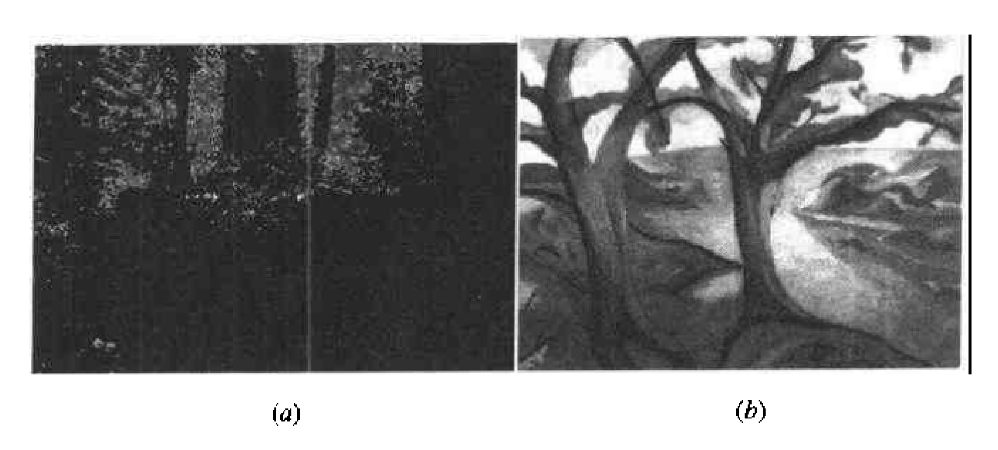
当用户使用这种方法时，创建的图形窗口初始化是空白的。如果用户使用 8 位显示系统，那么必须确保调色板入口的总数不超过 256。例如，如果用户试图显示三幅图像，每一幅都采用一个不同的 128 色调色板，那么至少有一幅图像将显示为错误的颜色(如果三幅

图像的调色板是一致的，那么将不会产生问题，因为只有 128 个颜色通道被使用）。注意，灰度图像总是使用调色板来进行显示的，所以这些图像所使用的颜色通道总数不能超过 256。

在以下介绍的例子中，有两幅索引图像同时显示在一个 8 位系统中。由于这些图像使用不同的调色板，并且同时受到屏幕颜色分辨率的限制，因而第一幅图像将被迫使用第二幅图像的调色板，从而导致不正确的显示结果：

```
[X1,map1]=imread('forest.tif');
[X2,map2]=imread('trees.tif');
imshow(X1,map1),figure,imshow(X2,map2);
```

显示效果如图 2.10(*a*)和图 2.10(*b*)所示。



(*a*)　(*b*)

图 2.10　使用同一个调色板的两幅图像的显示效果

(*a*) 调色板错误产生的图像显示效果；(*b*) 调色板正确产生的图像显示效果

△ 说明：

事实上本例真正的显示结果与用户计算机上打开的其他应用程序窗口个数和系统使用的颜色通道数目密切相关，因此读者看到的效果可能与图 2.10 有所不同。

△

避免这些显示问题出现的一种方法就是对调色板进行操作，使之使用较少的颜色。另外还有几种方法可以解决这一问题：例如，将图像转换为 RGB(真彩)格式再进行显示(这是因为 MATLAB 将自动使用抖动和颜色近似方法来显示 RGB 图像)；可以使用 ind2rgb 函数将索引图像转换为 RGB 图像：

```
imshow(ind2rgb(X,map))
```

或者简单地使用 cat 命令将一幅灰度图像显示为一幅 RGB 图像：

```
imshow(cat(3,I,I,I))
```

可以将多幅图像显示在同一个单独的图形窗口中，达到这一目的有两种方法：一种方法是联合使用 imshow 函数和 subplot 函数；另一种方法是联合使用 subimage 函数和 subplot 函数。

subplot 函数将一个图形窗口划分为多个显示区域。subplot 的调用格式如下：

```
subplot(m,n,p)
```

这种格式将图形窗口划分为 m×n 个矩形显示区域，并激活第 p 个显示区域。例如，

如果希望并排显示两幅图像，可使用以下语句：

```
[X1,map1]=imread('forest.tif');
[X2,map2]=imread('trees.tif');
subplot(1,2,1), imshow(X1,map2)
subplot(1,2,2), imshow(X2,map2)
```

显示结果如图 2.11 所示。

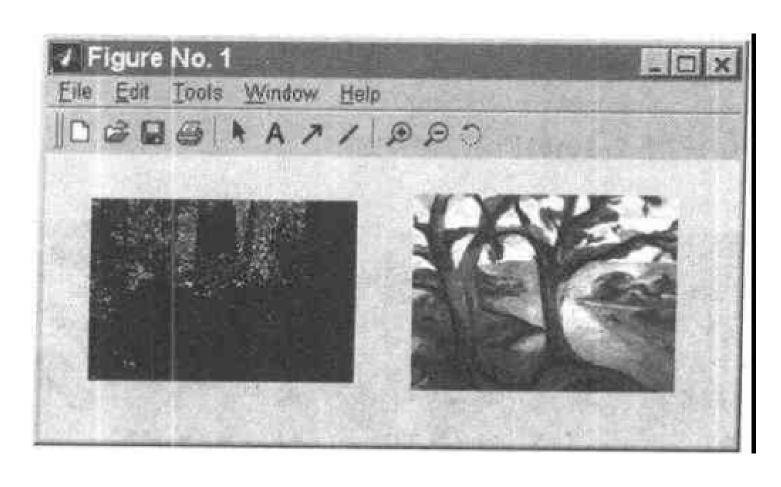



图 2.11　使用 imshow 函数同时显示两幅图像的效果

如果共享调色板(使用 subplot 函数)出现如图 2.11 所示的不可接受的显示结果，那么可以使用 subimage 显示函数其(其调用格式和 image 函数类似)，也可以在装载图像时将所有图像映射到同一个调色板中，这个调色板不是共享调色板情况下所采用的某一幅图像的调色板，而是映射后的包含所有图像调色板信息的一个新调色板(一般比每一个单独的图像调色板都要大一些)。subimage 函数在显示图像之前首先将图像转换为 RGB 图像，因此不会出现调色板问题。以下代码将显示与图 2.11 相同的两幅图像，但是效果要好得多：

```
[X1,map1]=imread('forest.tif');
[X2,map2]=imread('trees.tif');
subplot(1,2,1), subimage(X1,map1)
subplot(1,2,2), subimage(X2,map2)
```

显示结果如图 2.12 所示。

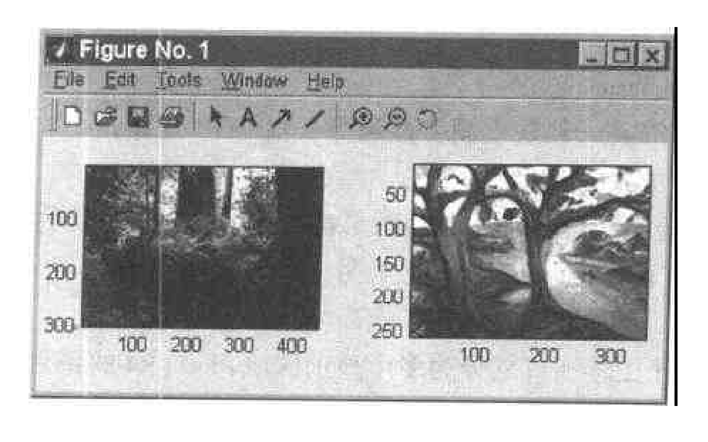



图 2.12　使用 subimage 函数同时显示两幅图像的效果

### 2.4.4 纹理映射

当用户使用 imshow 命令时，MATLAB 通常以二维视图形式显示一幅图像，我们也可以将一幅图像映射到一个参数化表面(例如，球体或曲面)上。warp 函数通过图像纹理映射创建这种三维显示效果。纹理映射使用插值方法将一幅图像映射到一个曲面网格上。warp 函数的调用形式如下：

```
warp (x,y,z,I)
```

其中，x,y,z 是可选参数，表示需要映射的表面形状(缺省时映射为一个简单矩形)，I 表示待映射的图像。以下代码将一幅测试图像纹理映射为一个圆柱：

```
[x,y,z] = cylinder;
I = imread('testpat1.tif');
warp(x,y,z,I);
```



图 2.13 图像纹理映射为圆柱形的效果

映射效果如图 2.13 所示。

有时图像可能不是按照用户所期望的形式进行纹理映射的，此时我们可以对纹理映射的外观进行修改，其方法之一就是修改坐标轴的 Xdir、Ydir 和 Zdir 属性值。

### 2.4.5 图像显示中的常见问题

**问题一**：彩色图像显示为灰度图像。用户图像可能是一幅索引图像，这就意味着显示这幅图像需要一个调色板。产生这个问题的原因可能是用户在装载索引图像时函数的调用方法不正确，正确的调用格式如下：

```
[X, map]=imread('filename.ext');
```

另外，还要注意使用 imshow 函数的正确形式：

```
imshow(X,map);
```

**问题二**：二值图像显示为全黑图像。使用 islogical 或 whos 命令检查该图像矩阵的逻辑标志是否置为 on。如果图像是逻辑的，那么 whos 命令将在类型头部单词 array 后面显示 logical。如果二值图像是自己创建的，那么产生这个问题的原因可能是图像类型为 uint8，记住 uint8 类型的灰度图像变化范围是[0, 255]，而不是[0, 1]。

**问题三**：装载的是多帧图像，但是 MATLAB 却仅仅显示一帧图像。用户必须单独装载多帧图像的每一帧，可以使用一个 for 循环来实现。用户可以先调用 imfinfo 函数获知图像帧数和图像维数。

## 【习　题】

1. 在一个单独的图像窗口中显示 MATLAB 图像处理工具箱中的任意一幅图像，然后利用窗口的菜单命令(窗口调节、坐标轴调节等)观察图像的缩放效果。

2. 将一幅世界地图变为一个类似地球仪的图形。

# 第三章 图像运算

**本章要点：**

★ 图像的点运算

★ 图像的代数运算

★ 图像的几何运算

★ 图像的邻域操作

## 3.1 图像的点运算

### 3.1.1 概述

点运算，也称为对比度增强、对比度拉伸或灰度变换，是一种通过对图像中的每个像素值(即像素点上的灰度值)进行计算，从而改善图像显示效果的操作。点运算常用于改变图像的灰度范围及分布，是图像数字化及图像显示的重要工具。在真正进行图像处理之前，有时可以用点运算来克服图像数字化设备的局限性。典型的点运算应用包括：

■ 光度学标定：通过对图像传感器的非线性特性作出补偿来反映某些物理特性，例如，光照强度、光密度等；

■ 对比度增强：调整图像的亮度、对比度，以便观察；

■ 显示标定：利用点运算使得图像在显示时能够突出所有用户感兴趣的特征；

■ 图像分割：为图像添加轮廓线，通常被用来辅助后续运算中的边界检测；

■ 图像裁剪：将输出图像的灰度级限制在可用范围内。

点运算是像素的逐点运算，它将输入图像映射为输出图像，输出图像每个像素点的灰度值仅由对应的输入像素点的灰度值决定。点运算不会改变图像内像素点之间的空间关系。设输入图像为 $A(x,y)$，输出图像为 $B(x,y)$，则点运算可表示为

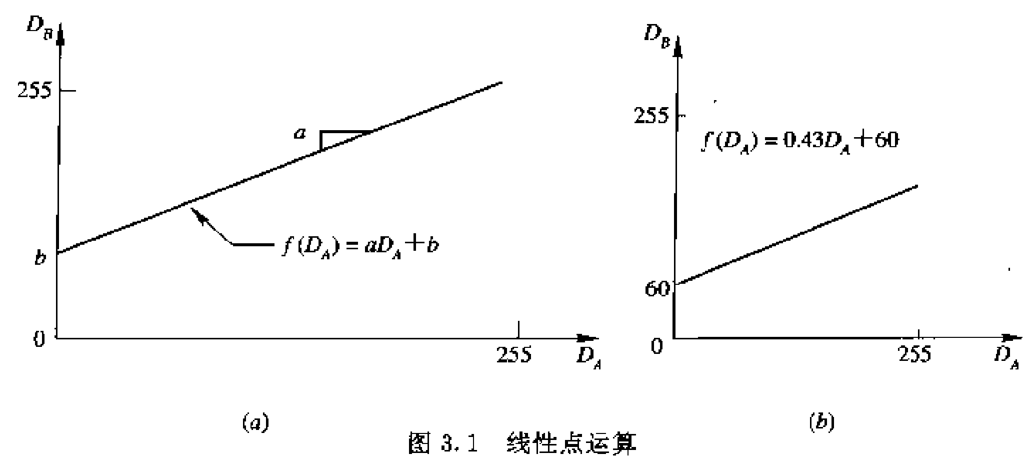
$$B(x,y) = f[A(x,y)]$$

点运算完全由灰度映射函数 $f$ 决定。根据 $f$ 的不同可以将图像的点运算分为线性点运算和非线性点运算两种。

### 3.1.2 线性点运算

线性点运算是指灰度变换函数 $f$ 为线性函数时的运算。用 $D_A$ 表示输入点的灰度值，$D_B$ 表示相应输出点的灰度值，则函数 $f$ 的形式如下(参见图 3.1(*a*))：

$$f(D_A) = aD_A + b = D_B \tag{3.1}$$

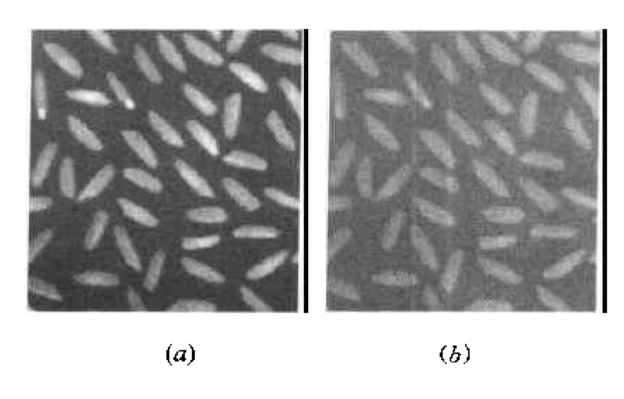



(a) (b)

图 3.1 线性点运算

(a) 线性变换函数曲线；(b) 减小对比度的线性变换函数曲线

当 $a>1$ 时，输出图像的对比度会增大；当 $a<1$ 时，输出图像的对比度会减小；当 $a=1$，$b=0$ 时，输出图像就是输入图像的简单复制；当 $a=1$，$b\neq 0$ 时，输出图像在整体效果上比输入图像要明亮或灰暗。例如，如果对图 3.2(a)所示的图像利用如图 3.1(b)所示的灰度变换函数进行点运算，那么将产生如图 3.2(b)所示的效果。



(a) (b)

图 3.2 图像对比度减小前、后的比较

(a) 原图像；(b) 对比度减小后的图像

除了调节图像的对比度以外，还有一种典型的线性点运算的应用就是灰度标准化。假如设灰度图像为 $I[W][H]$，其中 $W$ 表示图像宽度，$H$ 表示图像的高度，那么灰度图像的平均灰度和方差由如下计算公式得到：

平均灰度：

$$\bar{\mu}=\frac{1}{WH}\sum_{i=0}^{W-1}\sum_{j=0}^{H-1}D[i][j] \tag{3.2}$$

方差：

$$\bar{\sigma}^2=\frac{1}{WH}\sum_{i=0}^{W-1}\sum_{j=0}^{H-1}(D[i][j]-\bar{\mu})^2 \tag{3.3}$$

可以将其变换为均值和方差相同的灰度变换函数(线性映射)，其形式如下：

$$f(D[i][j])=\frac{\sigma_0}{\sigma}(D[i][j]-\bar{\mu})+\mu_0 \quad 0\leqslant i<W,\ 0\leqslant j<H \tag{3.4}$$

其中，$\sigma_0$ 和 $\mu_0$ 为给定的变换参数。灰度标准化可以用来生成一些常用的平均模型。

### 3.1.3 非线性点运算

非线性点运算对应于非线性的灰度变换函数。常用的非线性灰度变换函数包括平方函数、对数函数、窗口函数、阈值函数、多值量化函数等。图 3.3 给出了几种典型的非线性灰度变换函数曲线。

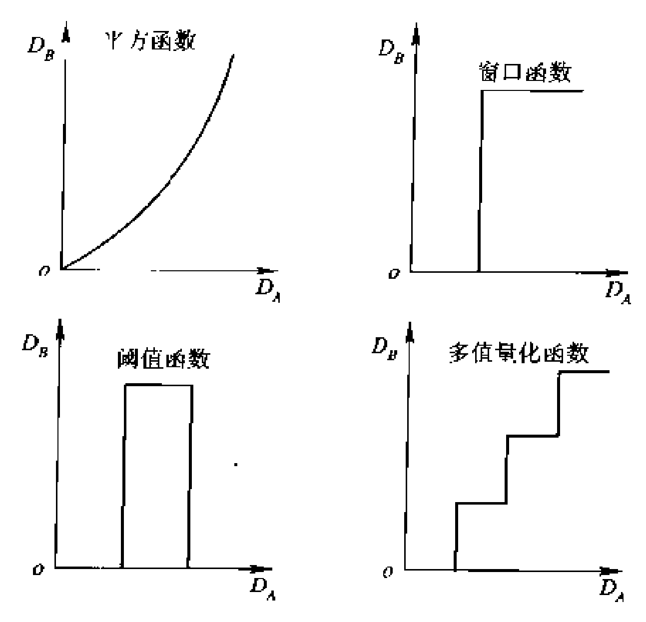



图 3.3　常用的非线性灰度变换函数

阈值化处理是最常用的一种非线性点运算，它的功能是选择一阈值，将图像二值化，然后使用生成的二进制图像进行图像分割及边缘跟踪等处理。利用阈值函数对图像进行非线性点运算，然后进行边缘检测的例子如图 3.4 所示(其中用到的边缘检测技术将在以后的章节中予以介绍)。

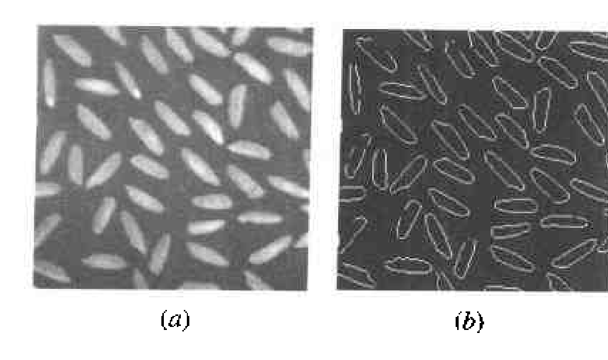



图 3.4　利用阈值函数进行边缘检测的显示效果
(a) 阈值化处理前；(b) 阈值化处理后

直方图均衡化也是一种常用的非线性点运算。直方图均衡化是指将一个已知灰度分布的图像使用某种非线性灰度变换函数进行计算，使运算结果变成一幅具有均匀灰度分布的新图像。如果对图 3.5(b)所示的图像使用图 3.5(a)所示的非线性灰度变换函数，那么就可以得到如图 3.5(c)所示的直方图均衡化效果。

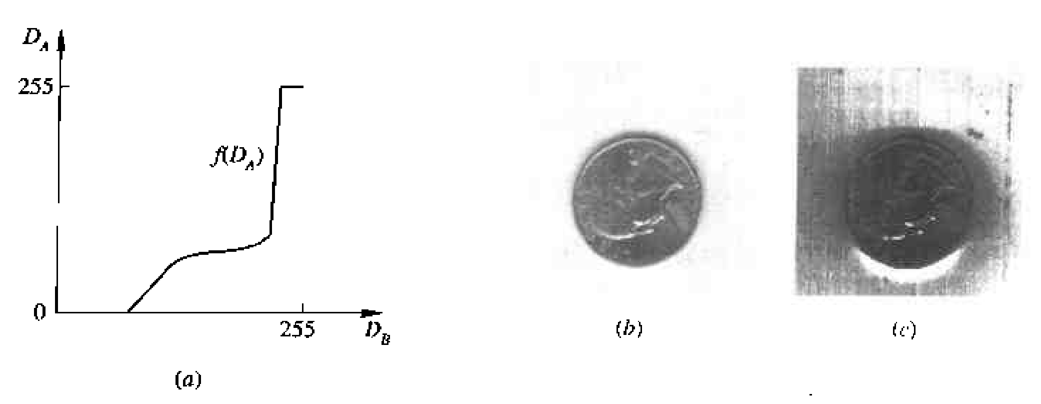
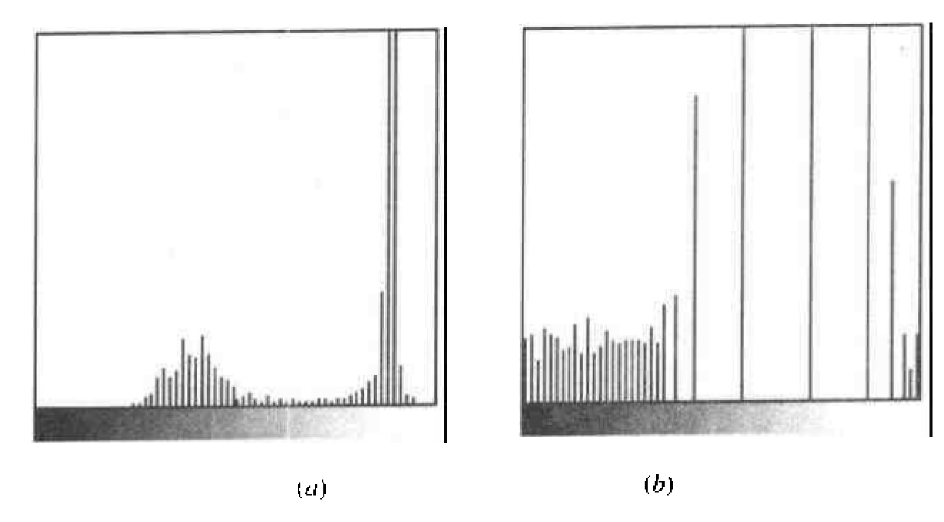



图 3.5　直方图均衡化前后的图像显示效果比较
(a) 非线性直方图均衡化函数；(b) 直方图均衡化前；(c) 直方图均衡化后

从图 3.6 中可以看出，经过均衡化后，原始图像的直方图被拉平了，但是经过直方图均衡化的点运算处理后，实际的直方图将呈现参差不齐的外观，这是由于输入图像可能的灰度级个数是有限的，而经过运算后某些灰度级可能没有像素，而某些灰度级则有许多像素。



(*a*) (*b*)

图 3.6 均衡化前、后的直方图比较

(*a*) 原图像直方图；(*b*) 均衡化后的图像直方图

### 3.1.4 MATLAB 的点运算实现方法

MATLAB 图像处理工具箱没有提供对图像进行直接点运算的函数，这是因为 MATLAB 图像处理工具箱着重于提供具有实际应用价值的函数，因而将图像的点运算过程直接集成在某些图像处理函数中(例如，直方图均衡化函数 histeq 和 imhist)。如果用户仅仅是希望对图像进行点运算处理，那么可以充分利用 MATLAB 强大的矩阵运算能力，对图像数据矩阵调用各种 MATLAB 计算函数进行处理。例如，假设希望对图 3.2(*a*)所示的灰度图像使用图 3.1(*b*)的灰度变换函数进行线性点运算，可以使用以下语句：

```
rice=imread('rice.tif');
I=double(rice);
J=I*0.43+60;
rice2=uint8(J);
subplot(1,2,1),imshow(rice)
subplot(1,2,2),imshow(rice2)
```

运算后得到的图像与图 3.2 所示的两幅图像是一致的。

**说明：**

由于 MATLAB 不支持 uint8 类型数据的矩阵运算，所以首先要将图像数据转换为双精度类型，计算完成后再将其转换为 uint8 类型。

# 3.2 图像的代数运算

## 3.2.1 概述

图像的代数运算是图像的标准算术操作的实现方法，是两幅输入图像之间进行点对点的加、减、乘、除运算后得到输出图像的过程。如果设输入图像为 $A(x,y)$ 和 $B(x,y)$，输出图像为 $C(x,y)$，则图像的代数运算有如下四种形式：

$$C(x,y) = A(x,y) + B(x,y)$$

$$C(x,y) = A(x,y) - B(x,y)$$

$$C(x,y) = A(x,y) * B(x,y)$$

$$C(x,y) = A(x,y)/B(x,y)$$

图像的代数运算在图像处理中有着广泛的应用，它除了可以实现自身所需的算术操作，还能为许多复杂的图像处理提供准备。例如，图像减法就可以用来检测同一场景或物体生成的两幅或多幅图像的误差。

我们可以使用 MATLAB 基本算术符(+、-、*、/等)来执行图像的算术操作，但是在此之前必须将图像转换为适合进行基本操作的双精度类型。为了更方便地对图像进行操作，图像处理工具箱包含了一个能够实现所有非稀疏数值数据的算术操作的函数集合。表 3.1 列举了所有图像处理工具箱中的图像代数运算函数。

**表 3.1 图像处理工具箱中的代数运算函数**

| 函数名 | 功能描述 |
|---|---|
| imabsdiff | 两幅图像的绝对差值 |
| imadd | 两个图像的加法 |
| imcomplment | 补足一幅图像 |
| imdivide | 两个图像的除法 |
| imlincomb | 计算两幅图像的线形组合 |
| lmmultiply | 两个图像的乘法 |
| imsubtract | 两个图像的减法 |

使用图像处理工具箱中的图像代数运算函数无需再进行数据类型间的转换，这些函数能够接受 uint8 和 uint16 数据，并返回相同格式的图像结果。虽然在函数的执行过程中元素是以双精度进行计算的，但是 MATLAB 工作平台并不会将图像转换为双精度类型。

代数运算的结果很容易超出数据类型允许的范围。例如，uint8 数据能够存储的最大数值是 255，各种代数运算尤其是乘法运算的结果很容易超过这个数值，有时算术操作(主要是除法运算)也会产生不能用整数描述的分数结果。图像的代数运算函数使用以下截取规则使运算结果符合数据范围的要求：超出数据范围的整型数据将被截取为数据范围的极值，分数结果将被四舍五入。例如，如果数据类型是 uint8，那么大于 255 的结果(包括无穷大 inf)将被设置为 255。

⚠ 注意：

无论进行哪一种代数运算都要保证两幅输入图像的大小相等，且类型相同。

### 3.2.2 图像的加法运算

图像相加一般用于对同一场景的多幅图像求平均效果(说明：此处的平均是指效果而言，并非算术平均)，以便有效地降低具有叠加性质的随机噪声。直接采集的图像品质一般都较好，不需要进行加法运算处理，但是对于那些经过长距离模拟通讯方式传送的图像(如卫星图像)，这种处理是必不可少的。

在 MATLAB 中，如果要进行两幅图像的加法，或者给一幅图像加上一个常数，可以调用 imadd 函数来实现。imadd 函数将某一幅输入图像的每一个像素值与另一幅图像相应的像素值相加，返回相应的像素值之和作为输出图像。imadd 函数的调用格式如下：

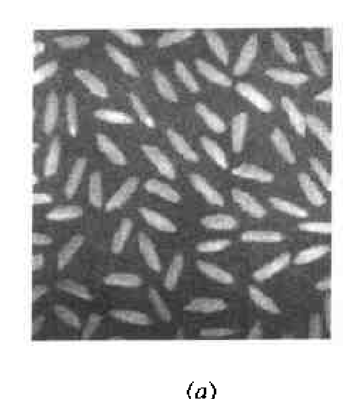
```
Z = imadd(X,Y)
```

其中，X 和 Y 表示需要相加的两幅图像，返回值 Z 表示得到的加法操作结果。

图像加法在图像处理中应用非常广泛。例如，以下代码段使用加法操作将图 3.7 中的(*a*)、(*b*)两幅图像叠加在一起：

```
I = imread('rice.tif');
J = imread('cameraman.tif');
K = imadd(I,J);
imshow(K)
```

叠加结果如图 3.8 所示。



(*a*)



(*b*)



图 3.7　待叠加的两幅图像

(*a*) 图像一；(*b*) 图像二



图 3.8　叠加后的图像效果

给图像的每一个像素加上一个常数可以使图像的亮度增加。例如，以下代码将增加图 3.9(*a*)所示的 RGB 图像的亮度(加亮后的结果如图 3.9(*b*)所示)。

```
RGB = imread('flowers.tif');
RGB2 = imadd(RGB,50);
subplot(1,2,1); imshow(RGB);
subplot(1,2,2); imshow(RGB2);
```



(a)



(b)

图 3.9　图像亮度增加前、后的显示效果比较
(a) 亮度增加前；(b) 亮度增加后

两幅图像的像素值相加时产生的结果很可能超过图像数据类型所支持的最大值，尤其是对于 uint8 类型的图像，溢出情况最为常见。当数据值发生溢出时，imadd 函数将数据截取为数据类型所支持的最大值，这种截取效果称之为饱和。为了避免出现饱和现象，在进行加法计算前最好将图像转换为一种数据范围较宽的数据类型。例如，在加法操作前将 uint8 图像转换为 uint16 类型。

### 3.2.3　图像的减法运算

图像减法也称为差分方法，是一种常用于检测图像变化及运动物体的图像处理方法。图像减法可以作为许多图像处理工作的准备步骤。例如，可以使用图像减法来检测一系列相同场景图像的差异。图像减法与阈值化处理的综合使用往往是建立机器视觉系统最有效的方法之一。在利用图像减法处理图像时往往需要考虑背景的更新机制，尽量补偿因天气、光照等因素对图像显示效果造成的影响。

在 MATLAB 中，使用 imsubtract 函数可以将一幅图像从另一幅图像中减去，或者从一幅图像中减去一个常数。imsubtract 函数将一幅输入图像的像素值从另一幅输入图像相应的像素值中减去，再将相应的像素值之差作为输出图像相应的像素值。imsubtract 函数的调用格式如下：

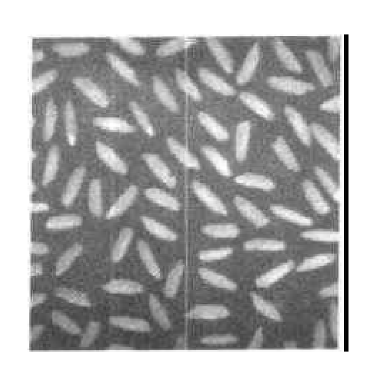
```
Z=imsubtract(X,Y)
```

其中，Z 是 X－Y 操作的结果。以下代码首先根据原始图像(如图 3.10(a)所示)生成其背景亮度图像，然后再从原始图像中将背景亮度图像减去，从而生成如图 3.10(b)所示的图像：

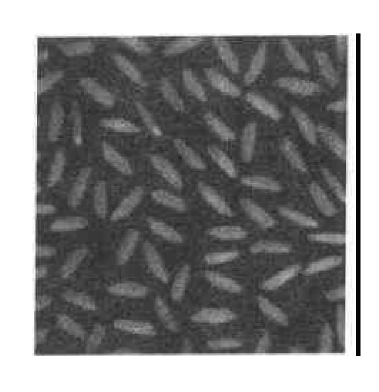
```
rice= imread('rice.tif');
background = imopen(rice, strel('disk',15));
rice2 = imsubtract(rice,background);
subplot(1,2,1); imshow(rice);
subplot(1,2,2); imshow(rice2);
```

如果希望从图像数据 I 的每一个像素中减去一个常数，可以将上述调用格式中的 Y 替换为一个指定的常数值，例如：

```
Z = imsubtract(I,50);
```



(a)



(b)

图 3.10　减去背景亮度前、后图像的显示效果比较
(a) 去除背景亮度前；(b) 去除背景亮度后

减法操作有时会导致某些像素值变为一个负数，对于 uint8 或 uint16 类型的数据，如果发生这种情况，那么 imsubtract 函数自动将这些负数截取为 0。为了避免差值产生负值，同时避免像素值运算结果之间产生差异，可以调用函数 imabsdiff。函数 imabsdiff 将计算两幅图像相应像素差值的绝对值，因而返回结果不会产生负数。该函数的调用格式与 imsubtract 函数类似。

### 3.2.4　图像的乘法运算

两幅图像进行乘法运算可以实现掩模操作，即屏蔽掉图像的某些部分。一幅图像乘以一个常数通常被称为缩放，这是一种常见的图像处理操作。如果使用的缩放因数大于 1，那么将增强图像的亮度，如果因数小于 1 则会使图像变暗。缩放通常将产生比简单添加像素偏移量自然得多的明暗效果，这是因为这种操作能够更好地维持图像的相关对比度。此外，由于时域的卷积或相关运算与频域的乘积运算对应，因此乘法运算有时也被作为一种技巧来实现卷积或相关处理。

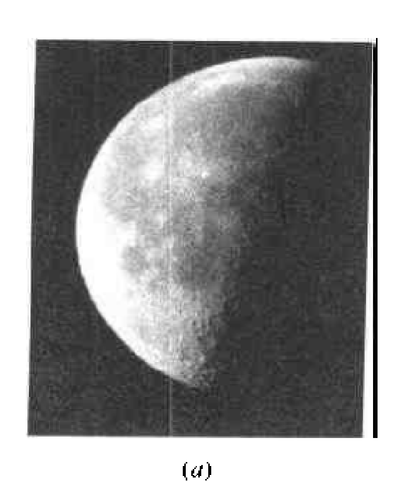
在 MATLAB 中，使用 immultiply 函数实现两幅图形的乘法。immultiply 函数将两幅图像相应的像素值进行元素对元素的乘法操作(MATLAB 点乘)，并将乘法的运算结果作为输出图像相应的像素值。immultiply 函数的调用格式如下：

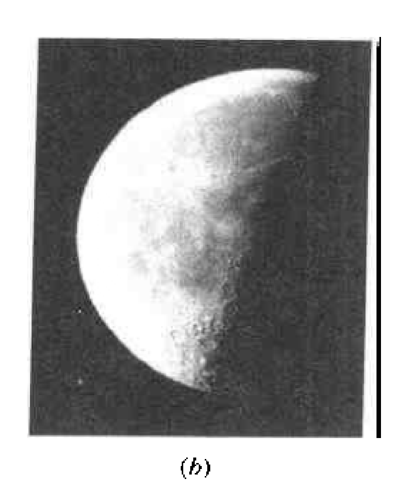
```
Z = immultiply(X,Y)
```

其中，Z=X＊Y。例如，以下代码将使用给定的缩放因数对图 3.11(a)所示的图像进行缩放，从而得到如图 3.11(b)所示的较为明亮的图像：

```
I = imread('moon.tif');
J = immultiply(I,1.2);
subplot(1,2,1); imshow(I);
subplot(1,2,2); imshow(J);
```

uint8 图像的乘法操作一般都会发生溢出现象。immultiply 函数将溢出的数据截取为数据类型的最大值。为了避免产生溢出现象，可以在执行乘法操作之前将 uint8 图像转换为一种数据范围较大的图像类型(例如，uint16)。





(a)　　(b)

图 3.11　图像缩放前、后的显示效果比较
(a) 缩放前；(b) 缩放后

### 3.2.5　图像的除法运算

除法运算可用于校正成像设备的非线性影响，这在特殊形态的图像(如断层扫描等医学图像)处理中常常用到。图像除法也可以用来检测两幅图像间的区别，但是除法操作给出的是相应像素值的变化比率，而不是每个像素的绝对差异，因而图像除法也称为比率变换。

在 MATLAB 中使用 imdivide 函数进行两幅图像的除法。imdivide 函数对两幅输入图像的所有相应像素执行元素对元素的除法操作(点除)，并将得到的结果作为输出图像的相应像素值。imdivide 函数的调用格式如下：

```
Z=imdivide(X,Y)
```

其中，Z=X/Y。例如，以下代码将图 3.10 所示的两幅图像进行除法运算，读者可以将这个结果与减法操作的结果相比较，对比它们之间的不同之处：

```
rice=imread('rice.tif');
I=double(rice);
J=I*0.43-90;
rice2=uint8(J);
Ip=imdivide(rice,rice2);
imshow(Ip,[])
```

除法操作的结果如图 3.12 所示。

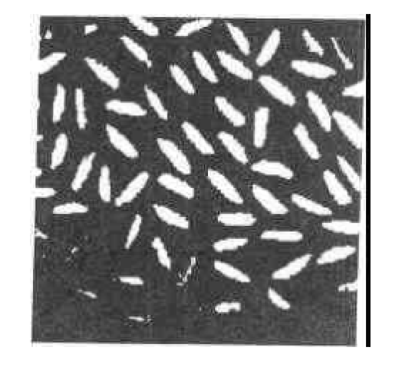


图 3.12　图像除法运算的显示效果

### 3.2.6　图像的四则代数运算

可以综合使用多种图像代数运算函数来完成一系列的操作。例如，使用以下语句计算两幅图像的平均值：

```
I = imread('rice.tif');
I2 = imread('cameraman.tif');
K = imdivide(imadd(I,I2), 2);
```

建议读者最好不要用这种方式进行图像操作，这是因为，对于uint8或uint16数据，每一个算术函数在将其输出结果传递给下一项操作之前都要进行数据截取，这个截取过程将会大大减少输出图像的信息量。执行图像四则运算操作较好的一个办法就是使用函数imlincomb。函数imlincomb按照双精度执行所有代数运算操作，而且仅对最后的输出结果进行截取，该函数的调用格式如下：

```
Z = imlincomb(A,X,B,Y,C)
```

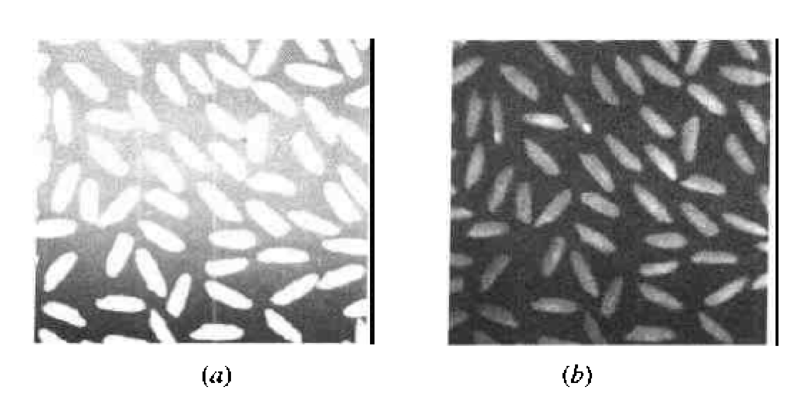
其中，Z=A * X + B * Y + C。MATLAB会自动根据输入参数的个数判断需要进行的运算。例如，以下语句将计算Z=A * X + C：

```
Z = imlincomb(A,X,C)
```

而以下语句将计算Z=A * X + B * Y：

```
Z = imlincomb(A,X,B,Y)
```

图3.13(*a*)、(*b*)分别给出了运用多个代数函数和使用imlincomb函数得到的不同四则运算结果。从图中可以看出，两种方法的结果有着显著的不同。



(*a*) (*b*)

图3.13 多个代数函数与imlincomb函数运算结果比较
(*a*)多个代数函数运算结果；(*b*)imlincomb函数运算结果

## 3.3 图像的几何运算

### 3.3.1 几何运算与坐标系统

几何运算与点运算不同，可以看成是像素在图像内的移动过程，该移动过程可以改变图像中物体对象(像素)之间的空间关系。几何运算可以是不受任何限制的，但是通常都需要做出一些限制以保持图像的外观顺序。完整的几何运算需要由两个算法来实现：空间变换算法和灰度插值算法。空间变换主要用来保持图像中曲线的连续性和物体的连通性，一般都采用数学函数形式来描述输入、输出图像相应像素间的空间关系。空间变换的一般定义为

$$g(x,y) = f(x',y') = f[a(x,y),b(x,y)]$$

其中，$f$表示输入图像，$g$表示输出图像，坐标$(x',y')$指的是空间变换后的坐标，要注意这时的坐标已经不是原来的坐标$(x,y)$了。$a(x,y)$和$b(x,y)$分别是图像的$x$和$y$坐标的空间变换函数。

灰度级插值主要是对空间变换后的像素赋予灰度值，使之恢复原位置处的灰度值。在几何运算中，灰度级插值是必不可少的组成部分，因为图像一般用整数位置处的像素来定义。而在几何变换中，$g(x,y)$的灰度值一般由处在非整数坐标上的$f(x,y)$的值来确定，即$g$中的一个像素一般对应于$f$中的几个像素之间的位置，反过来看也是一样，即$f$中的一个像素往往被映射到$g$中的几个像素之间的位置。

显然，要了解空间变换，首先就要对图像的坐标系统有一个清楚的了解。MATLAB 图像处理工具箱主要采用两种坐标系统：像素坐标系统和空间坐标系统。

描述位置最方便的方法就是使用像素坐标。在这种坐标系统下，图像被视为如图 3.14(*a*)所示的离散元素网格，网格按照从上到下、从左到右的顺序排列。

对于像素坐标来说，第一个分量$r$(行)向下增长，第二个分量$c$(列)向右增长。像素坐标是整型数值，数据范围在 1 到行或列长度之间。像素坐标与 MATLAB 矩阵下标一一对应，这种对应关系有助于理解图像数据矩阵与图像显示之间的关系。例如，图像第五行、第二列的像素值将保存在矩阵元素(5,2)中。在像素坐标中，一个像素被理解为一个离散单元，由一个单独的坐标对(例如，(5,2))惟一确定，根据这种定义方法，诸如(5.3,3.2)这样的位置是没有意义的。

在很多情况下，我们将像素视为一个正方形是非常有用的，此时坐标(5.3,3.2)是有意义的，并且该位置与坐标(5,2)是有区别的。图 3.14(*b*)说明了这种空间坐标系统的定义方法。

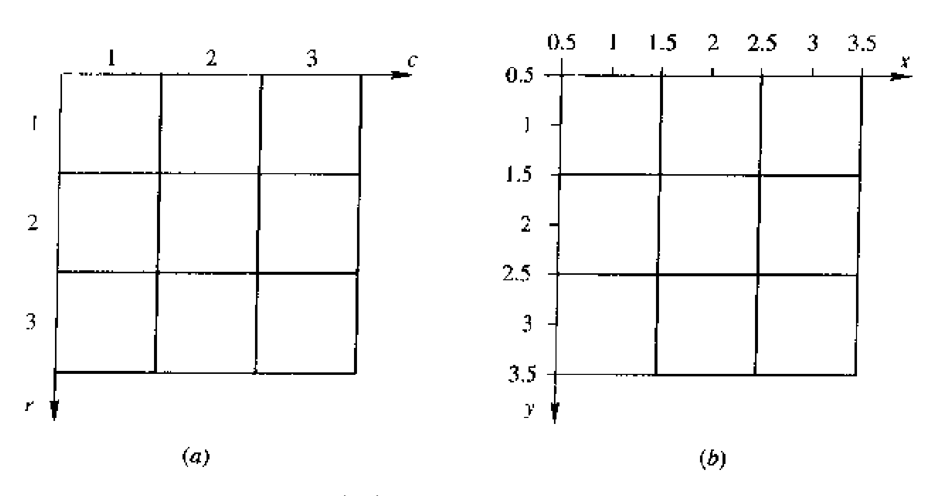



图 3.14　像素坐标系统和空间坐标系统

(*a*) 像素坐标系统；(*b*) 空间坐标系统

空间坐标系统在许多方面与像素坐标都是一致的。例如，任何像素中心点的空间坐标都与该像素的像素坐标一致。但是，这两种坐标系统也存在着很重要的区别：在像素坐标中，图像的左上角位置是(1,1)，而在空间坐标中，该位置缺省情况下为(0.5,0.5)。这是由于像素坐标系统是离散系统，而空间坐标系统是连续的；另外，在像素坐标系统中，左上角始终为(1,1)，但是在空间坐标系统中可以使用一个任意的起始点。例如，用户可以将图像的左上角指定为点(19.0,7.5)，而不是(0.5,0.5)。为了建立一个非缺省的空间坐标系统，可以在显示图像时指定图像的 XDATA 和 YDATA 属性，这两个属性都是由两个数值组成的向量，这两个数值分别表示第一个和最后一个像素的中心点坐标。例如，以下命令使用非缺省的 XDATA 和 YDATA 显示图像：

```
A = magic(5);
x = [19.5 23.5];
y = [8.0 12.0];
image(A,'XData',x,'YData',y), axis image, colormap(jet(25))
```

显示结果如图 3.15 所示。

缺省情况下，图像 A 的 XDATA 属性为[1,size(A,2)]，其中，size(A,2)表示矩阵 A 第二维的长度。而 YDATA 属性为[1,size(A,1)]，其中，size(A,1)表示矩阵 A 第一维的长度。显然，真实的坐标延伸范围略大于这两个数值间的距离。例如，如果 XDATA 为[1,200]，那么 x 轴图像延伸范围将为[0.5,200.5]。

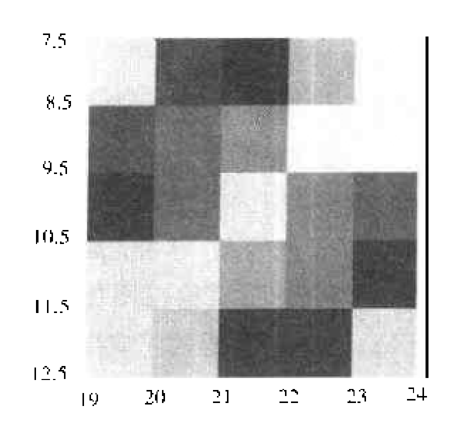



图 3.15　非缺省空间坐标系统下的图像显示效果

像素坐标与空间坐标另一个容易混淆的地方在于，两个坐标系统的水平分量符号和垂直分量符号是一种逆转关系，像素坐标从左到右表示图像列的方向，而空间坐标从左到右则相当于图像行的方向。在本书中，以 $r$ 和 $c$ 作为下标的函数采用的是像素坐标系统，而以 $x$ 和 $y$ 作为下标的函数采用的是空间坐标系统。

实现几何运算有两种方法：其一为前向映射法，即将输入像素的灰度一个个地转移到输出图像中，如果一个输入像素被映射到四个输出像素之间的位置，则其灰度值就按插值法在四个输出像素之间进行分配；其二为后向映射法(像素填充法)，这时将输出像素逐个地映射回输入图像中，若输出像素被映射到四个输入像素之间的位置，则其灰度由它们的插值来确定。在实际中，通常采用后向映射法。

几何变换常用于摄像机的几何校正过程，这对于利用图像进行几何测量的工作是十分重要的。本节将主要介绍图像匹配、改变图像大小、图像旋转、图像剪切、图像卷绕和图像扭曲这几种几何运算的概念及 MATLAB 的实现方法。

### 3.3.2　灰度级插值

灰度级插值是用来估计像素在图像像素间某一位置处取值的过程。例如，如果用户修改了一幅图像的大小，使其包含比原始像素更多的像素，那么必须使用插值方法计算其额外像素的灰度取值。

灰度级插值的方法有很多种，但是插值操作的方式都是相同的。无论使用何种插值方法，首先都需要找到与输出图像像素相对应的输入图像点，然后再通过计算该点附近某一像素集合的权平均值来指定输出像素的灰度值。像素的权是根据像素到点的距离而定的，不同插值方法的区别就在于所考虑的像素集合不同。例如，对于最近邻插值来说，输出像素将被指定为像素点所在位置处的像素值，其他像素都不考虑；对于双线性插值，输出像素值是像素 2×2 邻域内的权平均值；对于双三次插值，输出像素值是像素 4×4 邻域内的

权平均值。

最近邻插值是一种最简单的插值方法，但是这种方法有时不够精确。复杂一点的方法是双线性插值，该方法利用$(x,y)$点的四个最近邻像素的灰度值，按照以下方法计算$(x,y)$点的灰度值(参见图 3.16)。

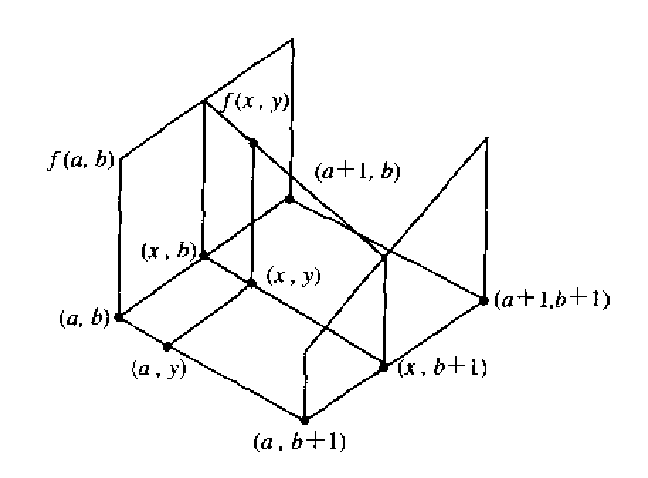



图 3.16 双线性插值示意图

假设，输出图像的宽度为 $W$，高度为 $H$，输入图像的宽度为 $w$，高度为 $h$，按照线形插值的方法，将输入图像的宽度方向分为 $W$ 等份，高度方向分为 $H$ 等份，那么输出图像中任意一点$(x,y)$的灰度值就应该由输入图像中四点$(a,b)$、$(a+1,b)$、$(a,b+1)$和$(a+1,b+1)$的灰度值来确定，$a$ 和 $b$ 的值分别为

$$a=\left[x\times\frac{w}{W}\right],\quad b=\left[y\times\frac{h}{H}\right]\qquad 0\leqslant x<W,\ 0\leqslant y<H \tag{3.5}$$

$(x,y)$点的灰度值 $f(x,y)$应为

$$f(x,y)=(b+2-y)f(x,b)+(y-b)f(x,b+1) \tag{3.6}$$

其中

$$f(x,b)=(x-a)f(a+1,b)+(a+1-x)f(a,b)$$

$$f(x,b+1)=(x-a)f(a+1,b+1)+(a+1-x)f(a,b+1)$$

更高阶数的插值通常是利用卷积来实现的，在实际应用中很少使用。MATLAB 图像处理工具箱可以使用三种插值方法：最近邻插值、双线性插值和双三次插值。每一个使用插值的几何运算函数都有一个用来指定插值方法的参数，大多数函数的默认方法是最近邻插值方法，这个方法对于所有图像类型都能够产生可接受的结果，并且是惟一一种可以用于索引图像的插值方法。使用最近邻插值方法产生的结果总是二进制的，因为插值将直接把输入图像的像素值作为输出图像的像素值。处理灰度和 RGB 图像可以指定使用双线性插值和双三次插值方法，因为这两种方法效果要好一些。对于 RGB 图像，插值将对红、绿、蓝三个颜色分量分别进行插值。

插值产生的输出图像的效果依赖于输入图像的类型：如果输入图像是双精度类型的，那么输出图像是一幅双精度类型的灰度图像。输出图像不是二值图像的原因是它包含的数值不全是 0 和 1；如果输入图像是 uint8 类型的，那么输出的是 uint8 类型的二值图像，插值得到的像素值被近似为 0 或 1，因此输出图像可以是 uint8 类型的。对于双三次插值，可能需要首先将双精度数据衰减到[0,1]范围内才能进行操作。

### 3.3.3 空间变换

空间变换是将输入图像的像素位置映射到输出图像的新位置处，常用的图像几何操作技术(例如，调整图像大小、旋转或剪切)都是空间变换的例子。空间变换既包括可用数学

函数表达的简单变换(例如，平移、拉伸等仿射变换)，也包括依赖于实际图像而不易用函数形式描述的复杂变换，例如，对存在几何畸变的摄像机所拍摄的图像进行校正，需要实际拍摄栅格图像，根据栅格的实际扭曲数据建立空间变换，再如通过指定图像中一些控制点的位移及插值方法来描述的空间变换。

仿射变换可以用以下函数来描述：

$$f(x) = Ax + b$$

其中，$A$ 是变形矩阵，$b$ 是平移矢量。在二维空间中，可以按照以下四种方式对图像应用仿射变换：

■ 尺度变换：选择变形矩阵为($s \geqslant 0$)

$$A_s = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \tag{3.7}$$

■ 伸缩：选择变形矩阵为

$$A_t = \begin{bmatrix} 1 & 0 \\ 0 & t \end{bmatrix} \tag{3.8}$$

■ 扭曲：选择变形矩阵为

$$A_u = \begin{bmatrix} 1 & u \\ 0 & 1 \end{bmatrix} \tag{3.9}$$

■ 旋转：选择变形矩阵为($0 \leqslant \theta \leqslant 2\pi$)

$$A_r = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{3.10}$$

对一幅图像分别进行以上四种仿射变换，产生的效果如图 3.17 所示。

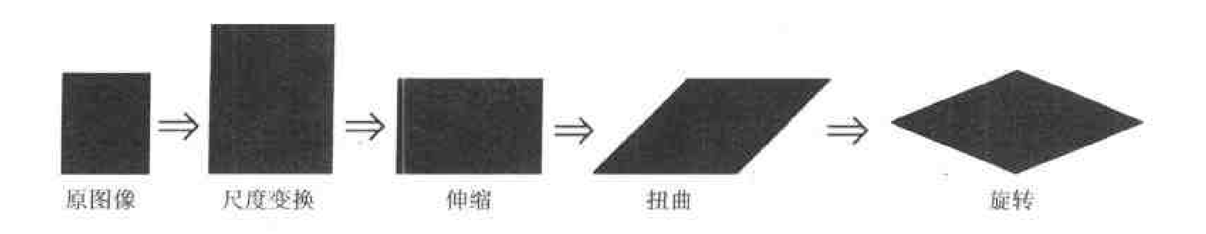



图 3.17 各种仿射变换的效果

最终的变形矩阵为

$$A_d = \begin{bmatrix} s\cos\theta & stu\cos\theta - st\sin\theta \\ s\sin\theta & stu\sin\theta + st\cos\theta \end{bmatrix} \tag{3.11}$$

另一种常用的空间变换方法是透视变换。透视变换是中心投影的射影变换，在用非齐次射影坐标表达时是平面的分式线性变换，常用于图像的几何校正。透视变换可以用以下形式进行描述：

$$x' = \frac{a_{11}x + a_{12}y + a_{13}}{a_{31}x + a_{32}y + a_{33}} \tag{3.12}$$

$$y' = \frac{a_{21}x + a_{22}y + a_{23}}{a_{31}x + a_{32}y + a_{33}} \tag{3.13}$$

其中，$a_{ij}$为指定的变换系数，且

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} \neq 0$$

MATLAB 进行空间变换的方法是：首先创建一个所需空间变换的结构体 TFORM，然后再调用 imtransform 函数，调用格式如下：

```
B = imtransform(A,TFORM,INTERP)
```

其中，A 表示需要变换的图像，INTERP 表示使用的插值方式，可以为'nearest'、'bilinear'或'bicubic'。除此之外，imtransform 函数还有一些用来控制变换外观的参数选项：Size 选项使变换图形看起来包含许多原始图像的拷贝；FillValues 选项用来控制位于输入图像边界以外的变换像素填充值；XYScale 选项用来指定输出像素的宽度和高度；XData 和 YData 选项用来指定输出图像在空间坐标中的位置；UData 和 VData 选项用来指定图像在像素坐标中的位置。TFORM 是一个变换结构体，可以使用 maketform 函数来创建。maketform 函数的基本调用格式如下：

```
T=maketform(type,A)
```

其中，A 表示需要变换的图像，type 是变换类型。表 3.2 列出了 maketform 函数所支持的变换类型。

**表 3.2　maketform 函数支持的空间变换类型**

| 变换类型 | 描　　述 |
|---|---|
| affine | 仿射变换，可能包括平移、旋转、尺度、拉伸和剪切等变换 |
| projective | 透视变换 |
| box | 对图像的每一维单独进行仿射变换的一种变换 |
| custom | 用户定义的变换 |
| composite | 两种或多种变换结合 |

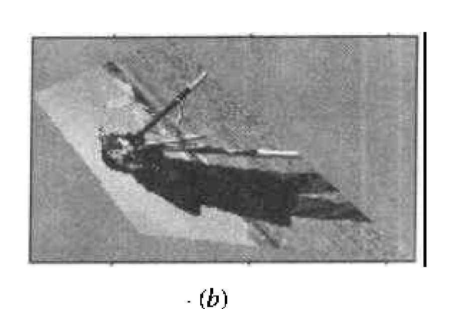
例如，以下语句可以得到一个仿射变换 TFORM 结构：

```
T = maketform('affine',[.5 0 0; .5 2 0; 0 0 1]);
```

创建了 TFORM 结构之后就可以调用 imtransform 函数来实现定义的空间变换。以下代码将图 3.18(a)所示的图像进行投影变换。

```
I = imread('cameraman.tif');
udata = [0 1]; vdata = [0 1];                    % 输入坐标系统
tform = maketform('projective',[ 0 0; 1 0; 1 1; 0 1],...
                  [-4 2; -8 -3; -3 -5; 6 3]);
[B,xdata,ydata] = imtransform(I,tform,'bicubic','udata',udata,...
                  'vdata',vdata,'size',size(I),'fill',128);
subplot(1,2,1), imshow(udata,vdata,I), axis on
subplot(1,2,2), imshow(xdata,ydata,B), axis on
```

以上代码将矩阵行[0 0;1 0;1 1;0 1]重新映射为行向量[−4 2;−8 −3;−3 −5;6 3]。变换前、后的图像如图 3.18(*a*)、(*b*)所示。

(*a*)　(*b*)

图 3.18　投影变换前、后图像的显示效果比较
(*a*) 投影变换前；(*b*) 投影变换后

这里要说明的一点是，在调用 imtransform 函数时，TFORM 结构必须定义为二维空间变换。如果一幅图像维数大于 2(例如，RGB 图像)，那么所有二维平面都将自动应用相同的二维变换。如果需要对任意维数的数组进行变换，可以使用 tformarray 函数。另外，综合使用 MATLAB 图像处理工具箱函数 maketform、fliptform、tformfwd、tforminv、findbounds、makeresampler、tformarray 和 imtransform 可以提供大量选项来定义并操作二维或 N 维空间变换。

### 3.3.4　几何运算的简单应用

**1. 改变图像大小**

MATLAB 使用 imresize 函数来改变一幅图像的大小。imresize 函数的调用格式如下：

```
B =imresize (A,M,METHOD)
```

其中，A 和 M 分别表示需要进行操作的图像和放大倍数，B 表示大小改变后的图像。如果希望放大一幅图像，可以指定一个大于 1 的放大倍数；如果希望缩小一幅图像，可以指定一个大于零、小于 1 的放大倍数。在缺省情况下，imresize 函数使用最近邻插值法，但是用户也可以通过指定 METHOD 参数来确定是使用双线性插值还是双三次插值方法。三种插值方法对应的 METHOD 的取值分别为'nearest'、'bilinear' 和'bicubic'。例如，以下命令将原图像放大 1.25 倍：

```
I = imread('ic.tif');
J = imresize(I,1.25);
imshow(I)
figure, imshow(J)
```

原图像与输出图像的对比如图 3.19 所示。

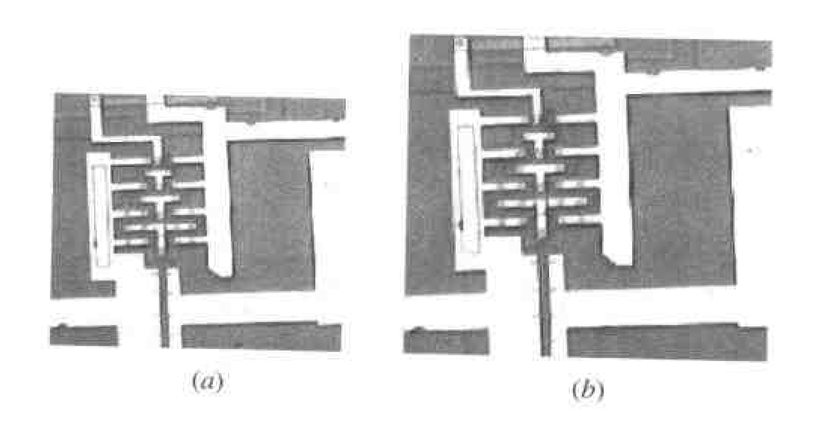


图 3.19　图像大小改变前、后的显示效果比较
(*a*) 改变前；(*b*) 改变后

调用 imresize 函数时可以指定输出图像的真实大小。例如，以下命令将创建一幅 100×150 的输出图像：

```
Y = imresize(X,[100 150])
```

⚠ 注意：

如果指定的大小不能够产生与输入图像同样的外观比例，那么输出的图像将会产生失真。

如果使用双线性插值或双三次插值来减小一幅图像的尺寸，那么 imresize 函数在进行插值之前将自动调用低通滤波器对图像进行滤波，这个滤波过程将减少重采样过程产生的波纹影响。然而，需要注意的是，即使使用了低通滤波器，改变大小的操作也会产生人为操作的痕迹，因为在改变图像大小时总会丢失部分信息。除非用户明确指定了滤波器，否则 imresize 函数在使用最近邻插值时不会使用低通滤波，因为这种插值方法主要用于索引图像，而低通滤波并不适合对索引图像进行操作。

**2. 图像旋转**

使用 imrotate 函数来旋转一幅图像。imrotate 函数的调用格式如下：

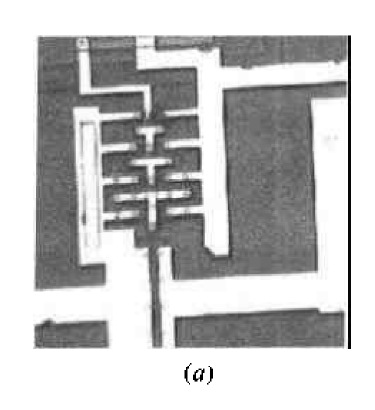
```
B = imrotate (A,ANGLE,METHOD,BBOX)
```

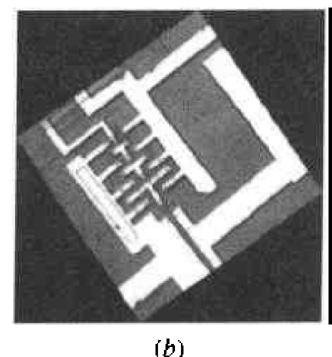
其中，A 表示需要旋转的图像，ANGLE 表示旋转的角度，METHOD 的取值与 imresize 函数一致。如果指定一个正的旋转角，那么 imrotate 函数将使用指定的插值方法和旋转角度将图像逆时针旋转；如果指定一个负值，那么将按顺时针方向旋转。这里要注意，由于旋转后的图像一般都不是一个矩形图像，所以 MATLAB 会自动将其填充为矩形图像：给输出图像外部缺少的像素赋值为 0，这将给输出图像产生一个黑色的背景，从而使得输出图像比原始图像要大一些。如果不特别说明(BBOX 取缺省值 loose)，那么 imrotate 函数返回的就是填充后的图像；如果指定参数 BBOX 为 crop，那么 imrotate 函数返回一幅剪裁过的、与原始图像同样大小的旋转图像。

例如，以下语句将分别显示两幅图像，它们分别是经逆时针旋转 35°后的未剪裁的和剪裁过的图像：

```
I = imread('ic.tif');
J = imrotate(I,35,'bilinear');
J1 = imrotate(I,35,'bilinear','crop');
subplot(1,3,1),imshow(I)
subplot(1,3,2), imshow(J)
subplot(1,3,3), imshow(J1)
```

旋转前与两幅旋转后的图像分别如图 3.20(*a*)、(*b*)、(*c*)所示。


(*a*)


(*b*)

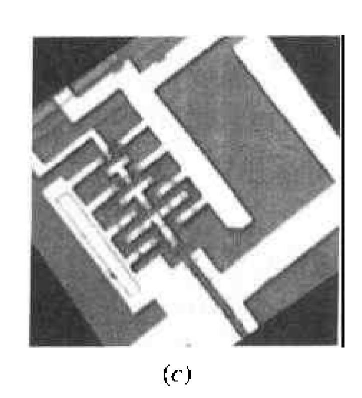

(*c*)

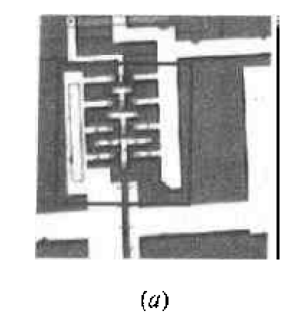
图 3.20 图像旋转前、后的显示效果比较
(*a*) 旋转前；(*b*) 旋转后未剪裁的图像；(*c*) 旋转后剪裁过的图像

**3. 图像剪切**

使用 imcrop 函数可以从一幅图像中抽取一个矩形的部分。imcrop 函数的调用格式如下：

```
X2 = imcrop(X,MAP,RECT)
```

其中，X 表示有待剪切的图像，不指定 X 时，imcrop 将当前坐标轴中的图像作为待剪切的图像。MAP 表示 X 为索引图像时的调色板，RECT 定义剪切区的矩形坐标。如果调用imcrop时不指定矩形的坐标，那么当光标位于图像中时会变成十字形，可以通过拖曳鼠标的方式交互式地选择一个矩形。imcrop 函数根据用户的选择绘制一个矩形，释放鼠标键后将产生一个新的图像。例如，首先显示一幅如图 3.21(*a*)所示的图像，然后调用 imcrop。imcrop函数会等待用户选择图像中的剪切区域，然后函数 imshow 将显示剪切得到的图像(假设用户选择的矩形如图 3.21 中的红色边框所示)：


(*a*)

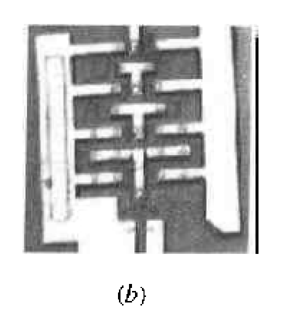

(*b*)

图 3.21 原图像以及图像矩形剪切部分的显示效果比较
(*a*) 原图像；(*b*) 剪切的矩形图像部分

```
imshow ('ic.tif')
I = imcrop;
imshow(I)
```

# 3.4 图像的邻域操作

## 3.4.1 概述

输出图像中的每个像素值都是由对应的输入像素及其某个邻域内的像素共同决定的，这种图像运算称为邻域运算。通常邻域是指一个远远小于图像尺寸的形状规则的像素块，例如，2×2、3×3、4×4的正方形，或用来近似表示圆及椭圆等形状的多边形。一幅图像所定义的所有邻域应该具有相同的大小。信号与系统分析中的相关与卷积基本运算，在实际的图像处理中都体现为某种特定的邻域运算。邻域运算与点运算一起形成了最基本、最重要的图像处理方法，尤其是滑动邻域操作，经常被用于图像的线性滤波和二值形态操作。

邻域操作包括两种类型：滑动邻域操作和分离邻域操作。在进行滑动邻域操作时，输入图像将以像素为单位进行处理，也就是说，对于输入图形的每一个像素，指定的操作将决定输出图像相应的像素值。分离邻域操作是基于像素邻域的数值进行的，输入图像一次处理一个邻域，即图像被划分为矩形邻域，分离邻域操作将分别对每一个邻域进行操作，求取相应输出邻域的像素值。

## 3.4.2 滑动邻域操作

滑动邻域操作一次处理一个像素，输出图像的每一个像素都是通过对输入图像某邻域内的像素值采用某种代数运算得到的。图3.22说明了一个6×5矩阵中3个元素的2×3滑动邻域，每一个邻域的中心像素都用一个黑点标出。

中心像素是输入图像真正要进行处理的像素。如果邻域含有奇数行和列，那么中心像素就是邻域的真实中心；如果行或列有一维为偶数，那么中心像素将位于中心偏左或偏上方。例如，在一个2×2的邻域中，中心像素就是左上方的像素，而图3.22所示的2×3邻域的中心像素为(1,2)，即位于邻域中第二列、第一行的像素。

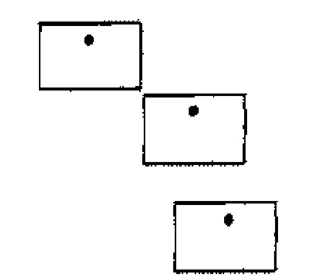


图3.22 6×5矩阵中3个元素的2×3滑动邻域

实现一个滑动邻域操作需要以下几个步骤：

(1) 选择一个单独的待处理像素。

(2) 判断像素的邻域。

(3) 对邻域中的像素值应用一个函数求值，该函数将返回标量计算结果。

(4) 找到输出图像与输入图像对应位置处的像素，将该像素的数值设置为上一步中得到的返回值。

(5) 对输入图像的每一个像素都重复步骤(1)到(4)。

例如，假设图3.22描述了一个均衡化操作。滑动邻域操作首先对邻域内的6个像素值求和，然后除以6，将结果作为邻域中心像素的取值。注意，邻域内某些像素很可能会丢失，尤其是当中心像素位于图像边界时。例如，在图3.22中，左上角和右下角的邻域将包

含不存在的像素，为了对这些包含了额外像素的邻域进行处理，滑动邻域操作将自动进行图像边界填充，通常使用数值 0 作为填充值，也就是说，滑动邻域操作中假定图像是由一些额外的全零行和全零列包围着，这些行和列不会成为输出图像的一部分，仅仅作为图像真实像素的临时邻域来使用。

可以使用滑动邻域操作来实现许多种类的滤波操作(例如，线性滤波的卷积操作)，这些操作在本质上都是非线性操作。例如，一个使输出图像像素值等于输入图像各个邻域像素值标准偏差的滑动邻域操作，可以使用 nlfilter 函数来实现多种滑动邻域操作。nlfilter 函数的调用格式如下：

```
B = nlfilter (A,'index',[M N],FUN,P1,P2,…)
```

其中，A 表示输入图像、[M N]指定邻域大小，FUN 是一个返回值为标量的计算函数，如果该计算函数需要参数，那么参数 P1、P2，……将紧跟在 FUN 参数后。返回值 B 是一个与输入图像相同大小的图像矩阵。index 是一个可选参数，如果存在该参数，那么 nlfilter 函数将图像 A 作为索引图像来处理。

以下语句将对图像 A 中每一个 3×3 的邻域调用 std2 函数(计算矩阵的标准方差)，结果返回一幅图像 B：

```
B = nlfilter(A,[3 3],'std2');
```

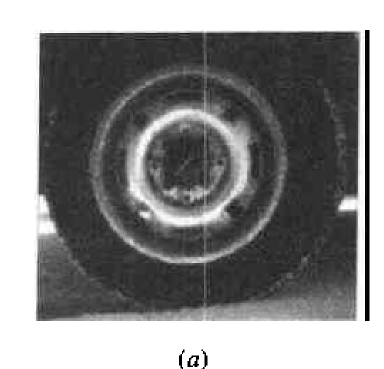
用户可以自己编写一个 M 文件来实现一个特定的计算函数，然后将该函数作为nlfilter 的参数。例如，以下命令将对矩阵 *I* 的 2×3 邻域使用用户自定义的名为 myfun.m 的函数：

```
nlfilter(I,[2 3],@myfun);
```

注意：调用中把函数句柄@myfun 作为参数。以下代码使用 nlfilter 函数将每一个像素值设置为 3×3 邻域内的最大值：

```
I = imread('tire.tif');
f = inline('max(x(:))');                % 构造复合函数 f=max(x(:))
I2 = nlfilter(I,[3 3],f);
subplot(1,2,1),imshow(I);
subplot(1,2,2),imshow(I2);
```

图像滑动邻域操作前后的效果如图 3.23 所示。



(*a*)



(*b*)

图 3.23 滑动邻域操作前、后图像显示效果比较
(*a*) 操作前；(*b*) 操作后

### 3.4.3　分离邻域操作

分离邻域是将矩阵划分为 m×n 后得到的矩形部分。分离邻域从左上角开始覆盖整个矩阵，邻域之间没有重叠部分。如果分割的邻域不能很好地适应图像的大小，那么需要为图像进行零填充。图 3.24 说明了一个被划分为 9 个 4×8 邻域的 11×22 矩阵，零填充过程将数值 0 添加到图像矩阵所需的底部和右边，此时图像矩阵大小变为 12×24。

函数 blkproc 可以实现分离邻域操作。blkproc 函数首先从图像中抽取一个分离邻域，然后将该邻域传递给指定的计算函数，最后由 blkproc 函数将返回的邻域组装起来创建一个输出图像。blkproc 函数的调用格式与 nlfilter 函数基本类似。例如，以下命令将对矩阵 ***I*** 中 4×6 大小的邻域使用函数 myfun 进行计算：

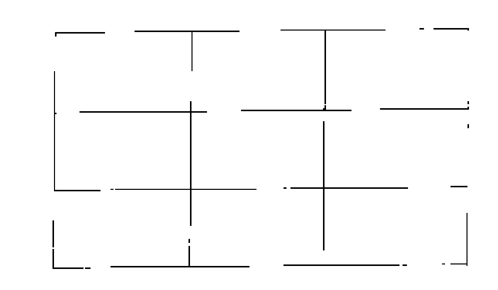


图 3.24　划分为 9 个 4×8 邻域的 11×22 矩阵

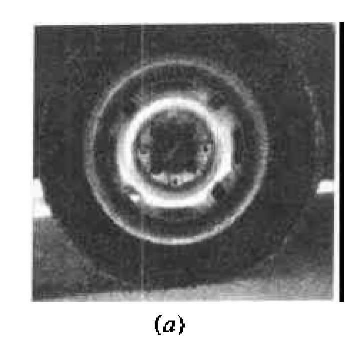
```
I2 = blkproc(I,[4 6],@myfun);
```

也可以指定 MATLAB 复合函数作为计算函数，例如：

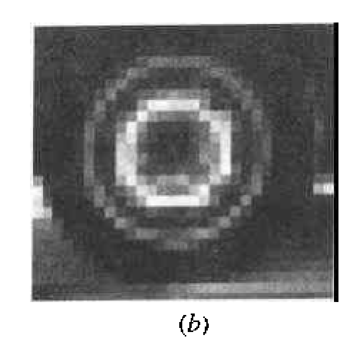
```
f = inline('mean2(x) * ones(s.ze(x))');      % 使用 inline 函数构造复合函数：
f = mean2(x) * ones(size(x))
I2 = blkproc(I,[4 6],f);
```

以下代码使用 blkproc 函数将输入图像(如图 3.25(*a*)所示)矩阵中 8×8 邻域的每一个像素值都设置为该邻域的平均值：

```
I = imread('tire.tif');
f = inline('uint8(round(mean2(x) * ones(size(x))))');
I2 = blkproc(I,[8 8],f);
subplot(1,2,1),imshow(I)
subplot(1,2,2),imshow(I2)
```



(*a*)



(*b*)

图 3.25　分离邻域操作前、后的显示效果比较

(*a*) 操作前；(*b*) 操作后

以上代码中的 inline 函数将首先计算邻域平均值，然后将输出结果与一个全 1 矩阵相乘，使生成的输出邻域与输入邻域具有相同的大小，从而保证输出图像与输入图象大小相同。虽然 blkproc 函数不要求输入、输出图像具有相同的大小，但是至少要确保函数能够返

回所需大小的邻域向量，而不是一个标量。计算后的结果如图 3.25(*b*)所示。从图中可以看出，由于同一个邻域中的像素取值都是一致的，所以很多细节内容都被删除了，图像的效果明显变差。从执行速度来看，分离块操作明显比滑动块操作快。

调用 blkproc 定义分离邻域时也可以使邻域之间相互重叠，在进行邻域处理时要重点考虑重叠部分的像素值。图 3.26 说明了一些在 15×30 矩阵中有 1×2 重叠部分的邻域，图中用阴影表示重叠的部分。

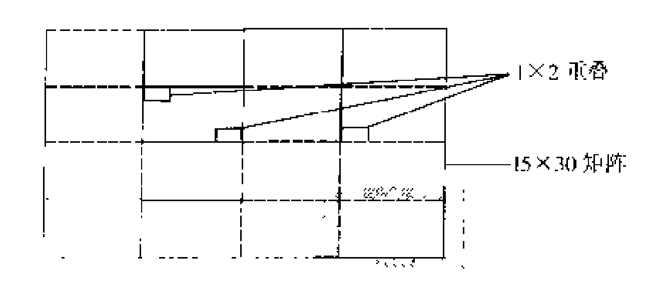



图 3.26 有重叠部分的分离邻域

如果存在重叠现象，那么 blkproc 函数将扩展后的邻域(包括重叠部分)传递给指定的函数。此时，blkproc 函数需要一个额外的输入参数来指定重叠部分的大小。例如：

```
B = blkproc(A,[4 8],[1 2],@myfun)
```

重叠往往会增加零填充过程的像素数目。例如，在图 3.26 中，原始的 15×30 的矩阵经过零填充后将变为一个 16×32 的矩阵，而如果原始矩阵的邻域有 1×2 的重叠部分时，填充后的矩阵将变为 18×36。图像最外面的矩形描述了为了适应重叠邻域处理进行零填充后产生的新边界。对图 3.25(*a*)所示图像进行有重叠的分离块操作，并利用减法操作比较有重叠和无重叠处理所得两幅图像的差别，代码如下：

```
I1=blkproc(I,[8 8],f);
I2 = blkproc(I,[8 8],[1 2],f);
I3=imsubtract(I2,I1);
imshow(I1)
imshow(I2)
imshow(I3)
```

从显示结果可以看出，两种操作的效果是完全一样的。

### 3.4.4 列处理

对于 MATLAB 语言来说，在进行图像处理之前将图像数据矩阵转换为矩阵列可以大大提高图像处理的速度。例如，假设当前进行的操作涉及到计算邻域的平均值，如果首先将邻域重新排列为列向量，那么计算过程将会更快，这是因为经过列处理后，可以通过调用一次 mean 函数对每一列(即每一个邻域)进行处理，无需再对每一个邻域单独调用一次 mean 函数。

可以使用 colfilt 函数来实现列处理操作。该函数的执行过程如下：首先将图像的每一个滑动或分离邻域重新排列到一个临时矩阵的某一列中，然后将临时矩阵传递给指定的计算函数，计算得到的结果经过重新排列变为原始图像的形状。colfilt 函数的调用格式如下：

```
B = colfilt(A,[M N],BLOCK_TYPE,FUN)
```

其中，A 为输入图像，[M N]为指定的邻域大小。BLOCK_TYPE 如果取值为'distinct'，则表示分离块操作；如果取值为'sliding'，则表示滑动块操作。FUN 为计算函数，B 为输出图像。以下语句将每一个输出像素的数值设置为输入像素邻域内的最大值，生成与 nlfilter 函数相同的输出结果：

```
I2 = colfilt(I,[3 3],'sliding',@max);
```

对于一个滑动邻域操作，原始图像中的每一个像素都对应于 colfilt 函数所创建的临时矩阵的一个单独列，该列包含该像素邻域内的所有数值。图 3.27(*a*)说明了临时矩阵列与图像像素间的对应关系，图中 6×5 的图像矩阵以 2×3 的邻域进行处理。colfilt 函数为每一个像素在临时矩阵中创建一个列，因此临时矩阵共有 30 列；每一个像素的列包含其邻域内的像素值，因此该矩阵有 6 行。如果有必要，colfilt 函数会对输入图像进行零填充。例如，图像左上角像素的邻域将会有两个 0 值相邻点，这就是零填充的结果。

临时矩阵必须传递给一个能够对每一列返回一个单独数值的函数(例如，mean、median、std、sum 等)，然后将返回的数值结果分配给输出图像相应的像素。colfilt 函数可以产生与 nifilter 函数相同的结果，但执行速度要快很多，它占用的内存要大于其他函数。

对于一个分离邻域操作而言，colfilt 函数通过将输入图像的每一个邻域进行重新排列来创建一个临时矩阵。在创建临时矩阵之前如果有必要的话会对原始图像进行零填充。图 3.27(*b*)说明了这个过程。使用 4×6 的邻域对 6×6 的矩阵进行处理，colfilt 函数首先对图像进行零填充，使之成为一个 8×18 的矩阵，然后将邻域重新排列为 6 列，每列包含 24 个元素。

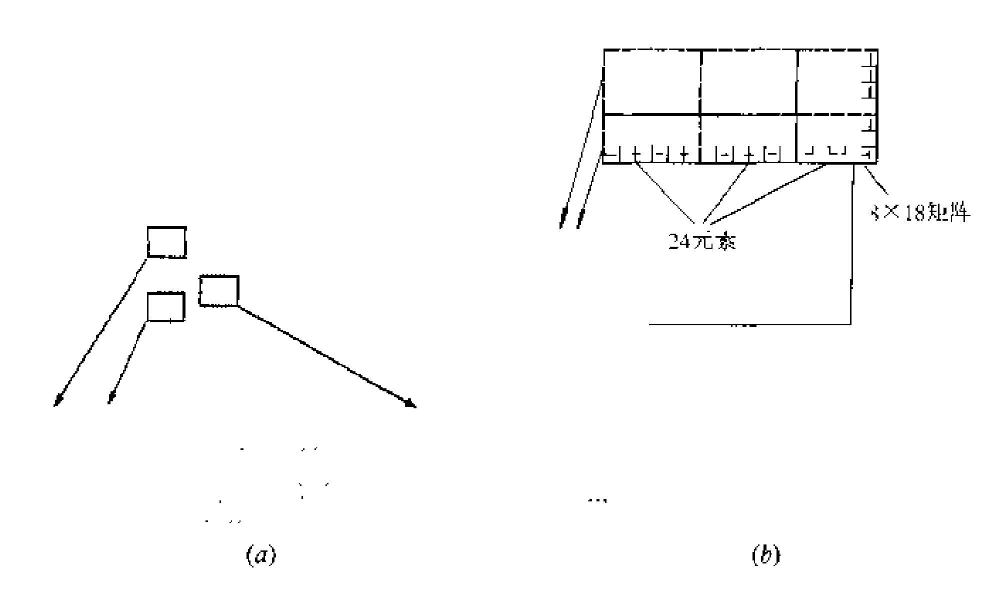



图 3.27　滑动邻域、分离邻域的临时矩阵列与像素的对应关系

(*a*) 滑动邻接操作；(*b*) 分离邻域操作

在将图像重新排列成为一个临时矩阵后，colfilt 函数将这个矩阵传递给指定的计算函数，计算函数必须返回与临时矩阵相同大小的矩阵结果。计算函数处理完临时矩阵后，输出将会重新排列为原始图像矩阵的形状。

虽然可以使用 colfilt 函数来实现一些与 blkproc 函数相同的分离邻域操作，但是 colfilt 函数有一些 blkproc 函数所没有的限制：输出图像与输入图像必须具有相同的大小；邻域之间不能重叠。对于不符合上述两个条件的情况，只能使用 blkproc 函数进行分离邻域操作。

## 【习 题】

1. 使用非线性灰度变换函数 $f(x)=1-x^2$ 对一幅彩色图像进行点运算，并与使用线性灰度变换函数 $f(x)=0.6x+4$ 进行点运算的结果相比较。

2. 编写一个程序以实现如下功能：将一个灰度图像与该图像少许平移后(边界全部填充为零)得到的图像相减后再相除，并显示和比较两种操作带来的不同的图像输出效果。

3. 使用列操作完成图 3.25 所示的分离邻域操作。

4. 实践图像交互式剪切方法。

# 第四章 图像变换

**本章要点：**

★ 傅立叶变换及其性质

★ 离散余弦变换

★ Radon 变换

★ 其他图像变换技术

## 4.1 傅立叶变换及其性质

### 4.1.1 概述

在计算机图像处理中，图像变换是一种为了达到某种目的(通常是从图像中获取某种重要信息)而对图像使用的一种数学技巧，经过变换后的图像将更为方便、容易地处理和操作。图像变换在图像处理中有着非常重要的地位，在图像增强、图像复原、图像编码压缩以及特征抽取方面有着广泛的应用。

从实际操作来看，图像变换就是对原图像函数寻找一个合适的变换核的数学问题。从本质上来说，图像变换有着深刻的物理背景。例如，函数的一次傅立叶变换反映了函数在系统频谱面上的频率分布。如果希望在频谱上作某些特定的处理，从而改变函数的某种特性(例如，图像增强)，那么可以再对函数作二次傅立叶变换。另外，图像经过一定的变换(傅立叶变换、离散余弦变换等)后，图像频谱函数的统计特性表明：图像的大部分能量都是集中在低、中频段的，高频分量很弱，仅仅体现了图像的某些细节。因此，可以通过图像变换来消除图像的高频段，从而达到图像压缩的目的。

在图像变换中，应用最广泛的变换就是傅立叶变换，从某种意义上说，傅立叶变换就是函数的第二种描述语言，掌握了傅立叶变换，人们就可以在空域或频域中同时思考处理问题的方法，这是一种非常重要的能力，有时能够使用简单的方法来解决非常复杂的问题。例如，空域中的函数卷积(参见 5.1 节内容)是一项比较复杂的运算，但是在频域中就转化为简单的函数乘法。除了傅立叶变换以外，还有一些其他很重要的图像变换方法，例如离散余弦变换、K－L 变换、Radon 变换等。

### 4.1.2 连续傅立叶变换

假设函数 $f(x)$ 为实变量 $x$ 的连续函数，且在 $(-\infty, +\infty)$ 内绝对可积，则 $f(x)$ 的傅立叶变换定义如下：

$$F(u)=\int_{-\infty}^{+\infty}f(x)e^{-j2\pi ux}dx \tag{4.1}$$

假设 $F(u)$可积，求 $f(x)$的傅立叶反变换定义如下：

$$f(x)=\int_{-\infty}^{+\infty}F(u)e^{j2\pi ux}du \tag{4.2}$$

式(4.1)和(4.2)称为傅立叶变换对。傅立叶变换前的变量域($x$)称为时域(或空域)，变换后的变量域($u$)称为频域。通常对这两个式子所做的假设在实际应用中都是成立的。从式(4.1)中可以看出，$F(u)$通常是自变量 $u$ 的复函数，可以将其表达为如下形式：

$$F(u)=R(u)+jI(u) \tag{4.3}$$

记作：

$$|F(u)|=\sqrt{R^2(u)+I^2(u)}$$

$$\theta(u)=\arctan\left[\frac{I(u)}{R(u)}\right]$$

称 $|F(u)|$为 $f(x)$的振幅谱或傅立叶谱，$\theta(u)$为傅立叶变换的相角。振幅谱的平方通常被称为 $f(x)$的能量谱。

傅立叶变换很容易推广到二维的情况，假设函数 $f(x,y)$是连续可积的，且 $F(u,v)$可积，则存在如下的傅立叶变换对：

$$F(u,v)=\int_{-\infty}^{+\infty}f(x,y)e^{-j2\pi(ux+vy)}dxdy \tag{4.4}$$

$$f(x,y)=\int_{-\infty}^{+\infty}F(u,v)e^{j2\pi(ux-vy)}dudv \tag{4.5}$$

同样可以将二维函数的傅立叶变换写为如下的形式：

$$F(u,v)=R(u,v)+jI(u,v) \tag{4.6}$$

振幅谱为

$$|F(u,v)|=\sqrt{R^2(u,v)+I^2(u,v)}$$

相角为

$$\theta(u,v)=\arctan\left[\frac{I(u,v)}{R(u,v)}\right]$$

### 4.1.3　离散傅立叶变换

在计算机上使用的傅立叶变换通常都是离散形式的，即离散傅立叶变换(DFT)。使用离散傅立叶类型变换的根本原因有二：一是 DFT 的输入、输出均为离散形式的，这使得计算机非常容易操作；二是因为计算 DFT 存在快速算法——快速傅立叶变换(FFT)，因而计算比较方便。

假设对函数 $f(x)$在 $N$ 个等间隔点处进行采样，得到离散化的函数 $f(m)$($m=1, 2, \cdots, N-1$)，定义一维离散傅立叶变换对形式如下：

$$F(p)=\frac{1}{N}\sum_{m=0}^{N-1}f(m)e^{-j\pi pm/N}\quad p=0,1,2,\cdots,N-1 \tag{4.7}$$

$$f(m)=\sum_{p=0}^{N-1}F(p)e^{-j\pi pm/N}\quad m=0,1,2,\cdots,N-1 \tag{4.8}$$

同样，在 $M\times N$ 的正方形网格上对函数 $f(x,y)$进行采样，可以得到二维离散化后的函数 $f(m,n)$。定义二维 DFT 和反 DFT 的关系如下：

$$F(p,q)=\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}f(m,n)\mathrm{e}^{-j(2\pi/M)pm}\mathrm{e}^{-j(2\pi/N)qn}\quad \begin{array}{l}p=0,1,\cdots,M-1\\ q=0,1,\cdots,N-1\end{array} \tag{4.9}$$

$$f(m,n)=\frac{1}{MN}\sum_{p=0}^{M-1}\sum_{q=0}^{N-1}F(p,q)\mathrm{e}^{j(2\pi/M)pm}\mathrm{e}^{j(2\pi/N)qn}\quad \begin{array}{l}m=0,1,\cdots,M-1\\ n=0,1,\cdots,N-1\end{array} \tag{4.10}$$

### 4.1.4 傅立叶变换的性质

傅立叶变换有以下这些非常重要的性质(详细描述见有关信号处理方面的书籍)：

■ 对称性：函数的偶函数分量将对应于傅立叶变换后的偶函数分量，奇函数分量也对应于奇函数分量，但是要引入系数－j；

■ 加法定理：时域中的加法对应于频域内的加法；

■ 位移定理：函数位移的变化不会改变其傅立叶变换的幅值，但会产生一个相位变化；

■ 相似性定理："窄"函数对应于一个"宽"傅立叶变换，"宽"函数对应于一个"窄"傅立叶变换(所谓的宽、窄是指函数在坐标轴方向上的延伸情况)；

■ 卷积定理：时域中的函数卷积对应于频域中的函数乘积；

■ 共轭性：将函数的傅立叶变换的共轭输入傅立叶变换程序得到该函数的共轭，也就是说，完全可以利用傅立叶变换程序计算傅立叶反变换而无须重新编写反变换程序；

■ Rayleigh 定理：傅立叶变换前、后的函数具有相同的能量。

对于二维傅立叶变换而言，还有以下两个特殊的重要性质：

■ 可分离性：如果二维函数可以分解为两个一维分量函数，那么傅立叶变换后的函数也可以分解为两个一维分量函数，这也就是说，对二维函数作傅立叶变换可以分为两步进行：首先视某一个方向变量为常数，对另一个方向作一维傅立叶变换，然后再对得到的变换结果作另一个方向上的一维傅立叶变换；

■ 旋转：如果函数在时域中旋转一个角度，那么其傅立叶变换也会旋转相同的角度。

傅立叶变换的这些性质在图像处理中有着广泛的应用，读者可以在今后的应用中充分体会到这一点。

### 4.1.5 图像的傅立叶变换

下面首先给出几个离散函数的傅立叶变换实例，使读者能够从中领会傅立叶变换的含义、性质和用途。

**例 4.1**：可视化离散函数 $f(m,n)$(参见图 4.1)的傅立叶变换结果。

函数 $f(m,n)$是一个在矩形区域内取值为 1，其他区域为 0 的离散函数(为了简化图像，图 4.1 将该函数表示为连续函数)。

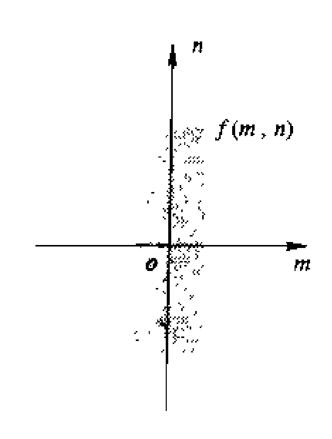



图 4.1 离散函数 $f(m,n)$示意图

图 4.2 说明了该函数的傅立叶变换幅值$|F(\omega_1,\omega_2)|$。幅值的网格图形绘制是傅立叶变换可视化的常用方法。

图 4.2 中心位置的尖峰表示 $F(0,0)$，此处的取值是函数 $f(m,n)$所有取值之和，通常称其为函数傅立叶变换的 DC 分量。注意，在 MATLAB 中，矩阵的下标是从 1 而不是从 0 开始的，所以矩阵元素 $F(1,1)$代表 $F(0,0)$，而 $f(1,1)$则代表了 $f(0,0)$。图 4.2 还说明，函数在水平频率上的能量比垂直频率处的能量要高，这正反映了傅立叶变换的相似性性质：函数在水平方向上是窄带脉冲，而垂直方向上是宽带脉冲，窄脉冲比宽脉冲的高频分量多一些。

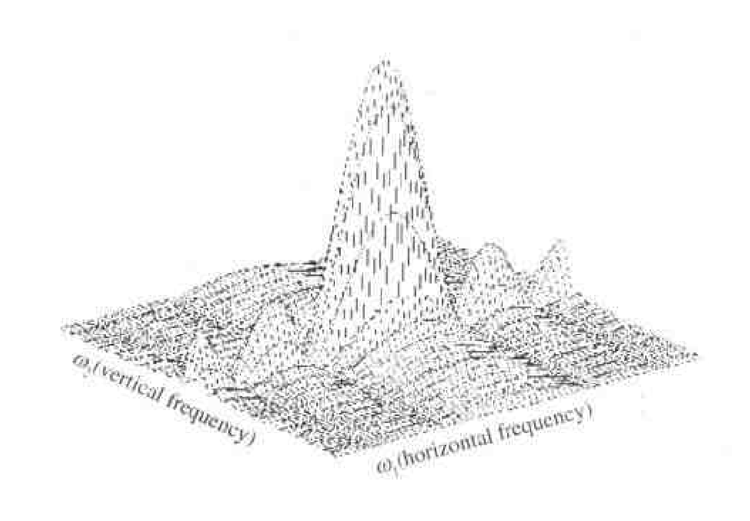



图 4.2 函数傅立叶变换幅值的网格图形

可视化函数傅立叶变换结果的另一个常用方法是将 $\lg|F(\omega_1,\omega_2)|$显示一幅图像，使用对数可以更详细地观察傅立叶变换在 0 附近区域的细节。本例的可视化结果如图 4.3 所示。

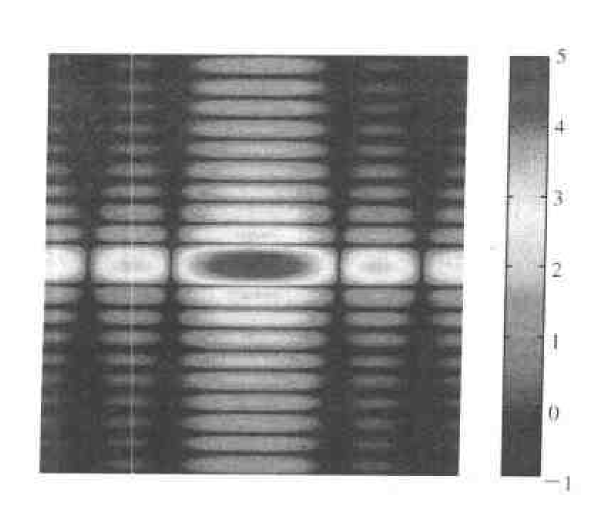



图 4.3 函数傅立叶变化幅值对数图形

**例 4.2**：典型图像的傅立叶变换。

图 4.4 给出了三幅典型的图像及其傅立叶变换结果。

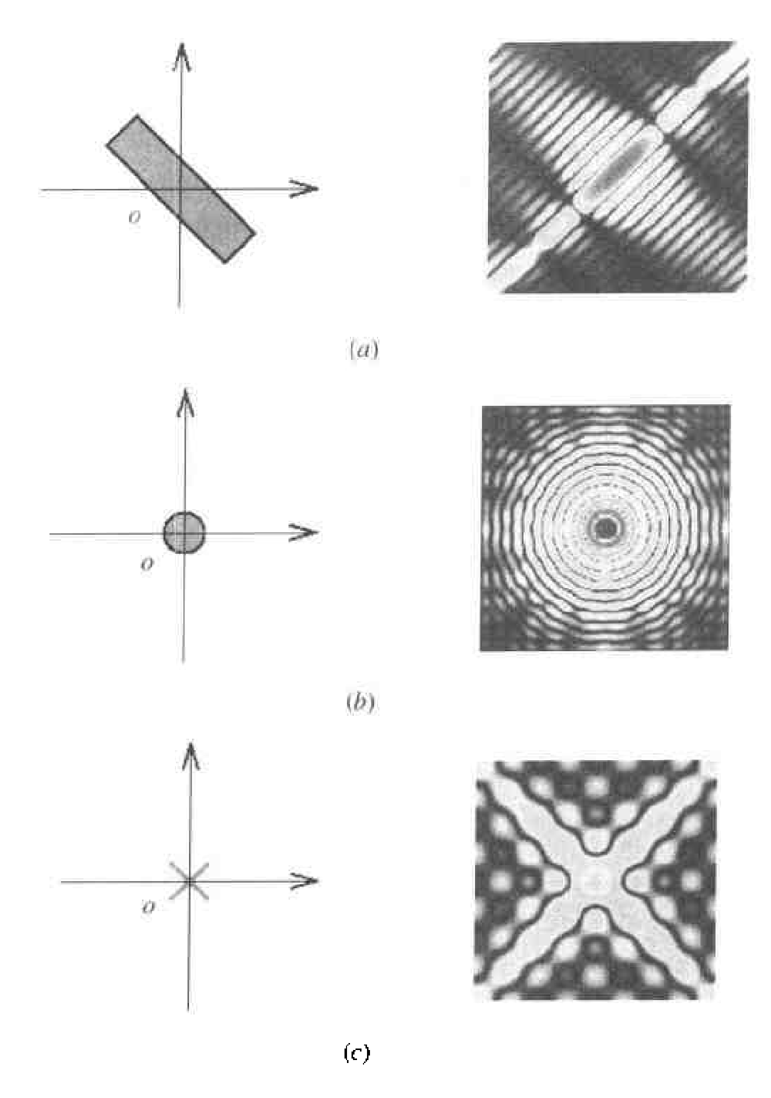



图 4.4　典型图像及其傅立叶变换结果

(a) 斜矩形图像及其傅立叶变换结果；(b) 圆形图像及其傅立叶变换结果；

(c) 叉形图像及其傅立叶变换结果

**例 4.3**：求线性滤波器的频率响应。

利用傅立叶变换可以得到线性滤波器的频率响应：首先求出滤波器的脉冲响应，然后利用快速傅立叶变换算法对滤波器的脉冲响应进行变换，得到的结果就是线性滤波器的频率响应。信号处理工具箱的 freqz2 函数就是利用这个原理计算滤波器的频率响应的。图 4.5 给出了使用 freqz2 函数得到的高斯滤波器的频率响应，程序代码如下：

```
h= fspecial('gaussian');
freqz2(h)
```

下面介绍如何利用 MATLAB 软件实现图像的傅立叶变换。MATLAB 函数 fft、fft2 和 fftn 分别可以实现一维、二维和 N 维 DFT 快速傅立叶变换算法，而函数 ifft、ifft2 和 ifftn 则用来计算反 DFT，它们是以需要进行反变换的图像作为输入参数，计算得到输出图像的。这些函数的调用格式如下：

```
A=fft(X,N,DIM)
```

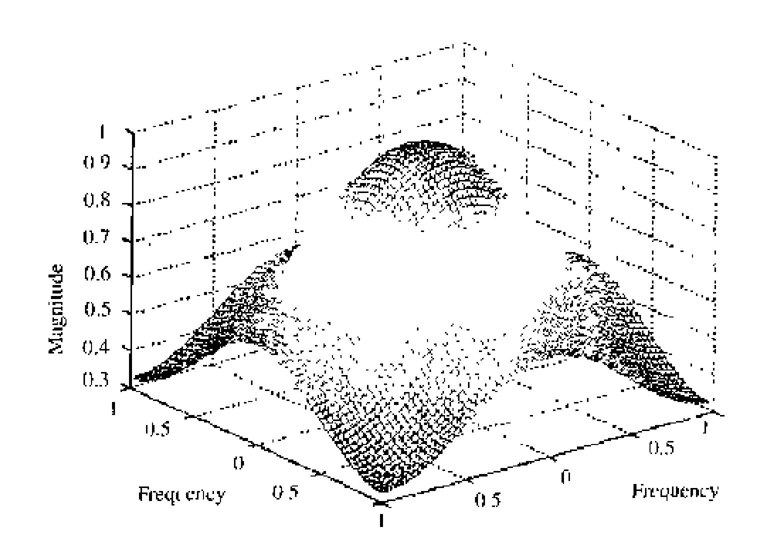



图 4.5 高斯低通滤波器

其中，X 表示输入图像【下同】。N 表示采样间隔点，如果 X 小于该数值，那么 MATLAB 将会对 X 进行零填充，否则将进行截取，使之长度为 N。DIM 表示要进行离散傅立叶变换的维数，A 为变换后的返回矩阵【下同】。

A=fft2(X,MROWS,NCOLS)

其中，MROWS 和 NCOLS 指定对 X 进行零填充后的 X 大小。

A=fftn(X,SIZE)

其中，SIZE 是一个向量，它们每一个元素都将指定 X 相应维进行零填充后的长度。

函数 ifft、ifft2 和 ifftn 的调用格式与对应的离散傅立叶变换函数一致。下面同样使用几个例子来说明傅立叶变换的实现方法和用途。

**例 4.4**：图像矩阵数据的显示及其傅立叶变换。

为了说明怎样根据图像矩阵数据进行图像的傅立叶变换，本例将构造一个类似于例 4.1 所示函数的矩阵 *f*，然后使用一个二进制图像来显示矩阵 *f*(数值 1 表示图 4.6 矩形的内部，0 表示其他位置)：

```
f = zeros(30,30);
f(5:24,13:17) = 1;
imshow(f,'notruesize')
```

使用以下命令计算并可视化 *f* 的 DFT 振幅谱：

```
F = fft2(f);
F2 = log(abs(F));
imshow(F2,[-1 5],'notruesize'); colormap(jet);
```

可视化结果如图 4.7 所示。

比较图 4.7 和图 4.3 所示的傅立叶变换结果，可以看出以下不同之处：首先，图 4.7 的傅立叶变换采样较为粗糙；其次，图 4.7 的零频率系数显示在图形的左上角，而不是传统的中心位置。

造成第一点不同的原因是快速傅立叶变换算法只能处理大小为 2 的幂次的矩阵(其他大小的矩阵可以采用其他非基 2 的混合基算法)，而本例中的矩阵维数并不是 2 的幂次。为

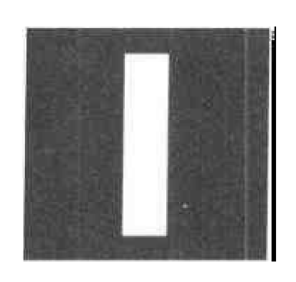


图 4.6 矩阵 $f$ 的二进制显示结果

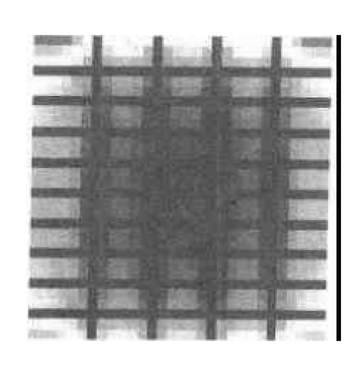


图 4.7 矩阵 $f$ 二进制图像的傅立叶变换结果

了解决这一问题，在计算 DFT 时可以通过对 $f$ 进行零填充来获得较好的傅立叶变换采样。零填充和 DFT 的计算可以使用以下语句完成。首先对 $f$ 进行零填充，得到一个 256×256 的矩阵，然后再计算 DFT 并显示其幅值谱：

```
F = fft2(f,256,256);
imshow(log(abs(F)),[-1 5]); colormap(jet);
```

其傅立叶变换结果如图 4.8 所示。

由图 4.8 所示，零频率系数仍然显示在图形的左上角而不是中心位置，这是因为，在计算图 4.1 所示函数的傅立叶变换时，坐标原点在函数图形的中心位置处，而计算机系统在执行傅立叶变换算法时是以图像的左上角为坐标原点的。我们可以使用函数 fftshift 对这个问题进行修正解决，该函数通过将图像 $F$ 的四个象限进行交换，使零频率系数位于图形的中心：

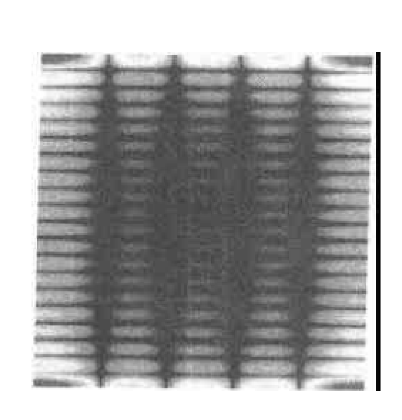


图 4.8 矩阵零填充后的傅立叶变换结果

```
F = fft2(f,256,256);
F2 = fftshift(F);
imshow(log(abs(F2)),[-1 5]); colormap(jet);
```

此时的绘图结果就会与图 4.3 相同了。

⚠ **说明**：

图像的四个象限是指以矩形图像的纵向以及对横向对称轴所分割的四个区域。

**例 4.5**：两个函数的快速卷积。

傅立叶变换的一个非常重要的属性就是两个傅立叶变换的乘积与相应空间函数的卷积相对应。假设 $\boldsymbol{A}$ 是一个 $M\times N$ 的矩阵，$\boldsymbol{B}$ 是一个 $P\times Q$ 的矩阵。$\boldsymbol{A}$ 与 $\boldsymbol{B}$ 的卷积可以由以下步骤计算得出：

(1) 对 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 进行零填充，将 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 填充为 2 的幂次矩阵。

(2) 使用 fft 计算 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的二维 DFT。

(3) 将两个 DFT 计算结果相乘。

(4) 使用 ifft2 计算步骤(3)所得的二维 DFT 的逆变换。

本例计算一个魔方阵和一个全1矩阵的卷积。

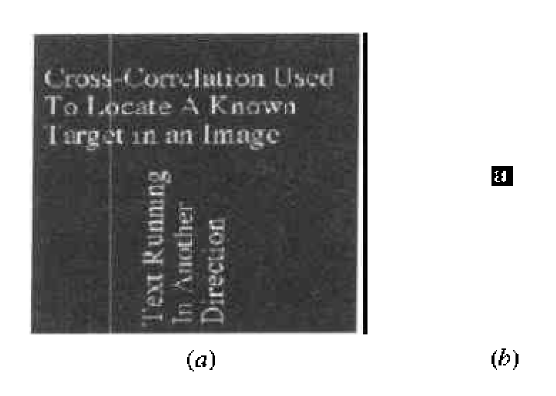
```
A = magic(3);
B = ones(3)
A(8,8) = 0;                    % 对A进行零填充，使之成为8×8矩阵
B(8,8) = 0;                    % 对B进行零填充，使之成为8×8矩阵
C = ifft2(fft2(A).*fft2(B));
C = C(1:5,1:5);                % 抽取矩阵中的非零部分
C = real(C)                    % 去掉错误的、由四舍五入产生的虚部
```

傅立叶变换结果如下：

```
C =
8.0000    9.0000    15.0000   7.0000    6.0000
11.0000   17.0000   30.0000   19.0000   13.0000
15.0000   30.0000   45.0000   30.0000   15.0000
7.0000    21.0000   30.0000   23.0000   9.0000
4.0000    13.0000   15.0000   11.0000   2.0000
```

**例 4.6**：定位图像特征。

傅立叶变换还能够用来分析两幅图像的相关性，相关性可以用来确定一幅图像的特征，在这个意义下，相关性通常被称为模板匹配。本例假设用户希望知道在一幅如图 4.9(*a*)所示的文本图像 text.tif 中有多少个字母 a。




(*a*) (*b*)

图 4.9 待分析的文本图像 text.tif 以及字母 a 的图像
(*a*) 文本图像；(*b*) 字母 a 的图像

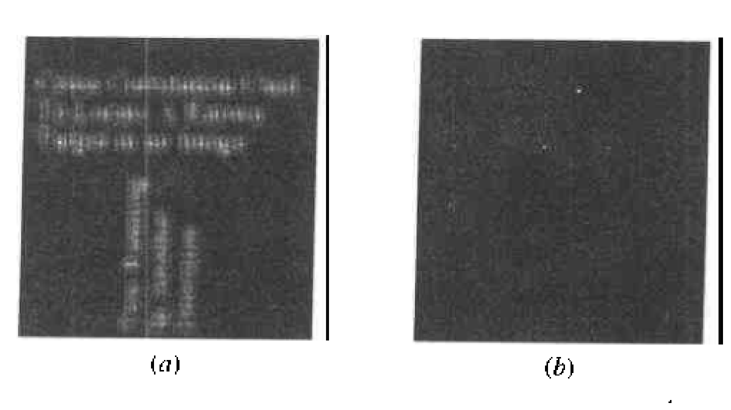
首先读入文件 text.tif，然后创建一个临时图像以便从中抽取字母 a：

```
bw = imread('text.tif');
a=bw(59:71,81:91);      % 从图像中抽取字母 a 的图像
subplot(1,2,1),imshow(bw);
subplot(1,2,2),imshwo(a);
```

字母 a 的临时图像如图 4.9(*b*)所示。将字母 a 图像旋转 180°后，计算图像 text.tif 与字母 a 图像的相关性就可以得到字母 a 的图像与大图像之间的关系。使用基于 FFT 的卷积技术来达到这一目的。为了使模板与图像相匹配，可以同时使用 fft2 和 ifft2 函数：

```
C = real(ifft2(fft2(bw) .* fft2(rot90(a,2),256,256)));
subplot(1,2,1),imshow(C,[])
thresh = max(C(:)) ;
% 找到 C 中的最大值，选择一个略小于该数的数值作为阈值
subplot(1,2,2),imshow(C > thresh)     %显示像素值超过阈值的像素
```

图 4.10(*a*)是相关性的计算结果，图中的亮点则相对于字母 a 出现的位置，这些点的位置由图 4.10(*b*)所示的极限相关图确认，说明图像 text.tif 中包含 5 个字母 a。



(*a*)　　(*b*)

图 4.10　卷积后的图像以及阈值化后的图像
(*a*) 卷积后的图像；(*b*) 阈值化后的图像

# 4.2　离散余弦变换

## 4.2.1　DCT 变换定义

离散余弦变换(DCT)是利用傅立叶变换的对称性，采用图像边界褶翻操作将图像变换为偶函数形式，然后对这样的图像进行二维离散傅立叶变换，变换后的结果将仅包含余弦项，故称之为离散余弦变换。DCT 可以将图像描述为不同幅值和频率的正弦值之和的形式。对于一幅典型的图像，DCT 有这样的性质：许多有关图像的重要可视信息都集中在 DCT 变换的一小部分系数中。因此，DCT 变化在图像压缩中非常有用，是有损图像压缩国际标准 JPEG 算法的核心。

一个 $M\times N$ 矩阵 $\boldsymbol{A}$ 的二维 DCT 定义如下：

$$B_{pq}=\alpha_p\alpha_q\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}A_{mn}\cos\frac{\pi(2m+1)p}{2M}\cos\frac{\pi(2n+1)q}{2N}\quad\begin{matrix}0\leqslant p\leqslant M-1\\0\leqslant q\leqslant N-1\end{matrix}\tag{4.11}$$

其中

$$\alpha_p=\begin{cases}\dfrac{1}{\sqrt{M}} & p=0\\[2ex]\sqrt{\dfrac{2}{M}} & 1\leqslant p\leqslant M-1\end{cases}$$

$$\alpha_q = \begin{cases} \dfrac{1}{\sqrt{N}} & q = 0 \\ \sqrt{\dfrac{2}{N}} & 1 \leqslant q \leqslant N-1 \end{cases}$$

数值 $B_{pq}$ 称为 **A** 的 DCT 系数。DCT 是一个可逆变换，逆变换的定义如下：

$$A_{mn} = \sum_{p=0}^{M-1}\sum_{q=0}^{N-1}\alpha_p\alpha_q B_{pq}\cos\frac{\pi(2m+1)p}{2M}\cos\frac{\pi(2n+1)q}{2N} \quad \begin{matrix} 0 \leqslant m \leqslant M-1 \\ 0 \leqslant n \leqslant N-1 \end{matrix} \tag{4.12}$$

其中

$$\alpha_p = \begin{cases} \dfrac{1}{\sqrt{M}} & p = 0 \\ \sqrt{\dfrac{2}{M}} & 1 \leqslant p \leqslant M-1 \end{cases}$$

$$\alpha_q = \begin{cases} \dfrac{1}{\sqrt{N}} & q = 0 \\ \sqrt{\dfrac{2}{N}} & 1 \leqslant q \leqslant N-1 \end{cases}$$

DCT 逆变换方程可以理解为：任意 $M\times N$ 的矩阵 **A** 都可以写成 $M\times N$ 个如式(4.13)所示的函数之和的形式：

$$\alpha_p\alpha_q B_{pq}\cos\frac{\pi(2m+1)p}{2M}\cos\frac{\pi(2n+1)q}{2N} \quad \begin{matrix} 0 \leqslant p \leqslant M-1 \\ 0 \leqslant q \leqslant N-1 \end{matrix} \tag{4.13}$$

这些函数被称为 DCT 基本函数。DCT 系数 $B_{pq}$ 可以看成是应用于每一个函数的权值。例如，对于一个 8×8 矩阵，它的 64 个基本函数如图 4.11 所示。

在图 4.11 中，这些基本函数的水平频率从左到右增长，垂直频率从上向下增长。位于图像左上方的定值基本函数通常被称为 DC 基本函数，相应的 DCT 系数称为 DC 系数。

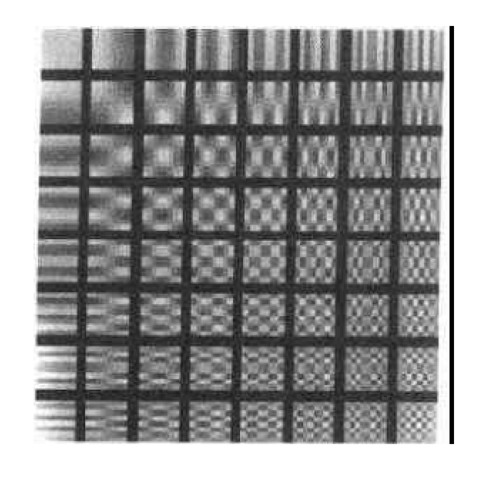


图 4.11 8×8 矩阵的 64 个基本函数

### 4.2.2 MATLAB 的 DCT 变换实现方法

$M\times M$ 变换矩阵 **T** 由下式给出：

$$T_{pq} = \begin{cases} \dfrac{1}{\sqrt{M}} & p = 0 \quad 0 \leqslant q \leqslant M-1 \\ \sqrt{\dfrac{2}{M}}\cos\dfrac{\pi(2q+1)p}{2M} & 1 \leqslant p \leqslant M-1 \quad 0 \leqslant q \leqslant M-1 \end{cases} \tag{4.14}$$

对于一个 $M\times M$ 矩阵 **A**，**T** * **A** 是一个 $M\times M$ 矩阵，该矩阵的列包含矩阵 **A** 列的一维 DCT。**A** 的二维 DCT 可以通过计算 **B**=**T** * **A** * **T'** 获得。由于 **T** 是一个实标准正交矩阵，所以其逆变化的形式与变换形式一致，因此，**B** 的二维逆 DCT 由 **T'** * **B** * **T** 给出。

图像处理工具箱提供两种不同的方法计算 DCT：第一种方法是使用函数 dct2、该函数使用一个基于 FFT 的算法来提高当输入较大的输入方阵时的计算速度。dct2 函数的调用格式如下：

```
B = dct2(A,[M N])
```

或

```
B = dct2(A,M,N)
```

其中，A 表示要变换的图像，M 和 N 是可选参数，表示填充后的图像矩阵大小。B 表示变换后得到的图像矩阵。

第二种方法使用由函数 dctmtx 返回的 DCT 变换矩阵，这种方法较适合于较小的输入方阵(例如，8×8 或 16×16 方阵)。dctmtx 的调用格式如下：

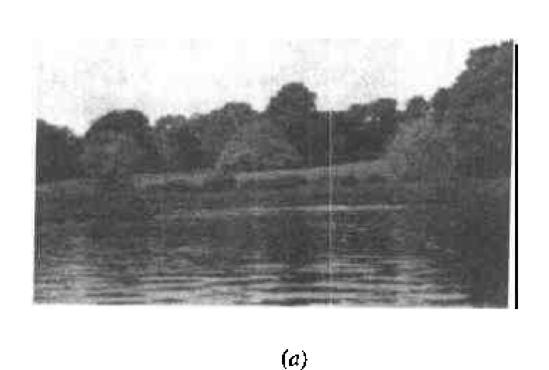
```
D = dctmtx(N)
```

其中，N 表示要变换的矩阵，D 为变换后得到的图像矩阵。该函数的使用方法参见 4.2.3 节的内容。下面给出一个例子说明 DCT 变换的实现方法。

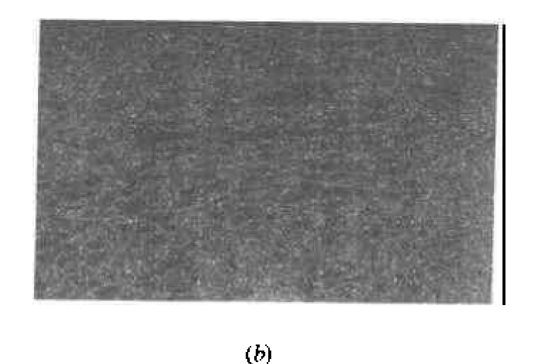
**例 4.7**：使用 dct2 函数计算图像的 DCT 变换。

以图 4.12(*a*)为例，调用 dct2 函数计算其 DCT 并显示结果(如图 4.12(*b*)所示)，程序代码如下：

```
RGB = imread('autumn.tif');
I = rgb2gray(RGB);
J = dct2(I);
subplot(1,2,1),imshow(I);
subplot(1,2,2),imshow(log(abs(J)),[]),colormap(jet(64))
```



(*a*)



(*b*)

图 4.12　离散余弦变换前、后图像显示效果比较
(*a*) 原始图像；(*b*) DCT 变换结果

### 4.2.3　DCT 和 JPEG 初步

DCT 是 JPEG 压缩算法的基础，以下将对 DCT 在 JPEG 中的应用做一个非常简单的介绍，详细内容可参见本书第八章。在 JPEG 图像压缩算法中，输入图像被分为 8×8 或 16×16 块，然后对每一个块计算二维 DCT，接着再对 DCT 系数进行量化、编码和发送。JPEG 接收者通过对 DCT 系数进行解码，计算每个块的二维逆 DCT，然后将所有的块重新组装成一幅图像。对于一般的图像来说，DCT 系数许多都是接近于 0 的数值，可以丢弃这

些系数而不会对图像的重建质量产生重大影响。

以下代码计算输入图像(如图 4.13(*a*)所示)的 8×8 个块的二维 DCT，然后丢弃块中那些近似于 0 的数值，只保留 64 个 DCT 系数中的 10 个，最后对每一个块使用二维逆 DCT 重新构造图像(如图 4.13(*b*)所示)。

```
I = imread('cameraman.tif');
I = im2double(I);
T = dctmtx(8);
B = blkproc(I,[8 8],'P1*x*P2',T,T');  %T和T转置是DCT函数P1*x*P2的参数【下同】
mask = [1 1 1 1 0 0 0 0
        1 1 1 0 0 0 0 0
        1 1 0 0 0 0 0 0
        1 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0];
B2 = blkproc(B,[8 8],'P1.*x',mask);
I2 = blkproc(B2,[8 8],'P1*x*P2',T',T);
subplot(1,2,1),imshow(I)
subplot(1,2,2),imshow(I2)
```

**说明：**

“.*”表示矩阵对应元素向量乘法，“*”表示矩阵乘法，二者有很大不同，使用哪一种乘法应视情况而定。





(*a*)



(*b*)

图 4.13　JPEG 压缩前、后显示效果比较

(*a*) 压缩前；(*b*) 压缩后

对比图 4.13 中的两幅图像可以看出，虽然几乎 85% 的 DCT 系数都被丢弃，导致重建的图像有一些质量损失，但是图像仍然是清晰可辨的。

# 4.3 Radon 变 换

## 4.3.1 Radon 变换

Radon 变换是计算图像在某一指定角度射线方向上投影的变换方法。Radon 变换与一种常用的计算机视觉操作 Hough 变换有着很密切的关系，通常会综合使用这两种变换进行图像分析。例如，可以使用 radon 函数实现某一种 Hough 变换，从而检测图像中的直线。

我们知道，二维函数 $f(x,y)$的投影是其在确定方向上的线积分。例如，$f(x,y)$在垂直方向上的二维线积分就是 $f(x,y)$在 x 轴上的投影；$f(x,y)$在水平方向上的二维线积分就是 $f(x,y)$在 y 轴上的投影。图 4.14 说明了一个简单二维函数的水平和垂直投影。

可以沿任意角度 $\theta$ 计算函数的投影，这也就是说，任意角度上都存在函数的 Radon 变换。图 4.15 说明了 Radon 变换的几何原理。

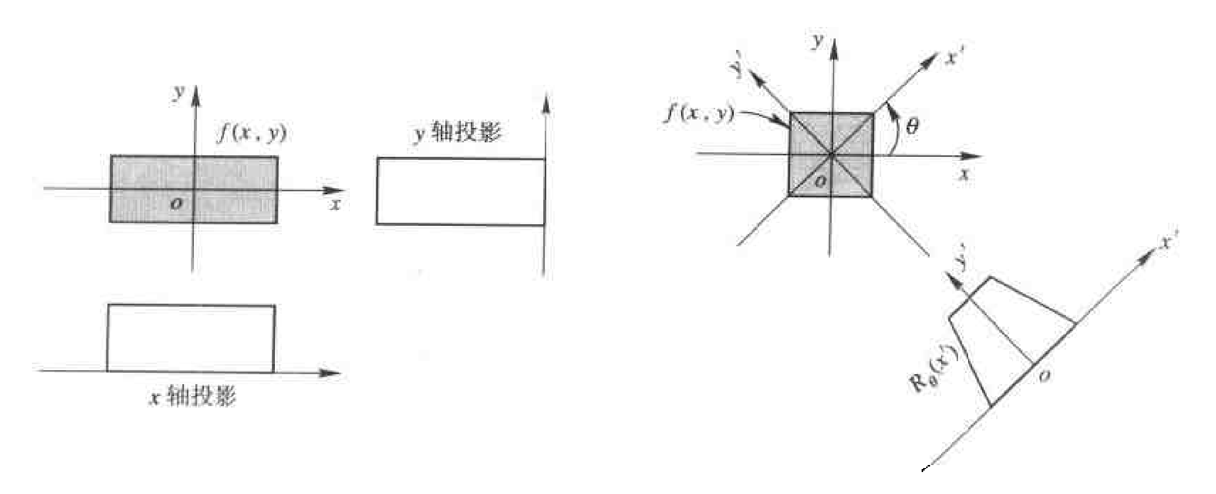



图 4.14 函数的水平和垂直投影示意图　　图 4.15 Radon 变换的几何原理示意图

通常情况下，$f(x,y)$的 Radon 变换是一个平行于 y 轴的线积分：

$$R_\theta(x') = \int_{-\infty}^{\infty} f(x'\cos\theta - y'\sin\theta,\ x'\sin\theta - y'\cos\theta)\mathrm{d}y' \qquad (4.15)$$

其中

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

## 4.3.2 MATLAB 的 Radon 变换实现方法

MATLAB 图像处理工具箱中的 radon 函数可以用来计算图像在指定角度的 Radon 变换。radon 函数的调用格式如下：

```
[R xp] = radon(I,theta,N)
```

其中，I 表示需要变换的图像。theta 表示变换的角度。N 是一个可选参数，指定 Radon 变换将在 N 点上进行计算，缺省条件下投影点的数目由下式决定：

```
2 * ceil(norm(size(I)-floor((size(I)-1)/2)-1))+3
```

返回值 R 表示的列里包含了对应于 theta 中每一个角度的 Radon 变换结果，如果指定了参数 N，那么 R 将包含 N 个行向量。向量 xp 包含相应的沿 x 轴的坐标。

大角度范围的 Radon 变换通常都会显示为一幅图像。例如，以下代码将计算正方形图像在 0°～180°范围内，以 1°增加的多个 Radon 变换(变换结果如图 4.16 所示)：

```
theta = 0:180;
[R,xp] = radon(I,theta);
imagesc(theta,xp,R);
set(gca,'XTick',0:20:180);
colormap(hot);
```

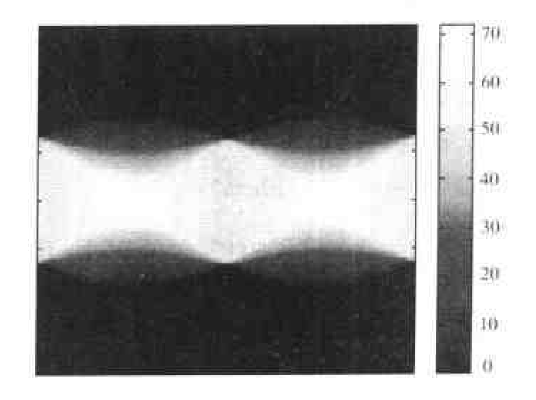


图 4.16 正方形从 0°～180°渐变的 Radon 变换的显示效果

### 4.3.3 Radon 逆变换

Radon 逆变换可以根据投影数据重建图像，在 X 射线断层摄影分析中常常使用。MATLAB 图像处理工具箱函数 iradon 可以实现 Radon 逆变换。给出一幅图像 I 和一个角度集合 theta，使用函数 radon 计算 Radon 变换：

```
R = radon(I,theta);
```

然后调用函数 iradon 重建图像 I：

```
IR = iradon(R,theta);
```

在以上代码中，投影是根据原始图像 I 计算得到的，而在大多数应用领域中是找不到构成投影的原始图像的。例如，在 X 射线吸收断层摄影中，投影是通过测量放射线沿不同角度穿透物理标本的衰减程度而构造出来的。原始图像可以理解为穿透标本的射线交叉部分，用灰度值表示标本的密度。使用特殊的硬件将投影图像收集起来，然后使用iradon 函数重新构造一个内部图像，这样能够实现对生物或不透明物体进行非侵害性照相。

iradon 函数可以使用平行投影束重新构造图像。图 4.17 说明了平行束几何是如何应用于 X 射线吸收断层摄影中的。此时发射器和接收器的数量是相同的，每一个检测器测量相应发送器的发送射线，射线的衰减程度可以由积分所得的物体密度或质量求得，这就相当于 Radon 变换中计算的线积分。

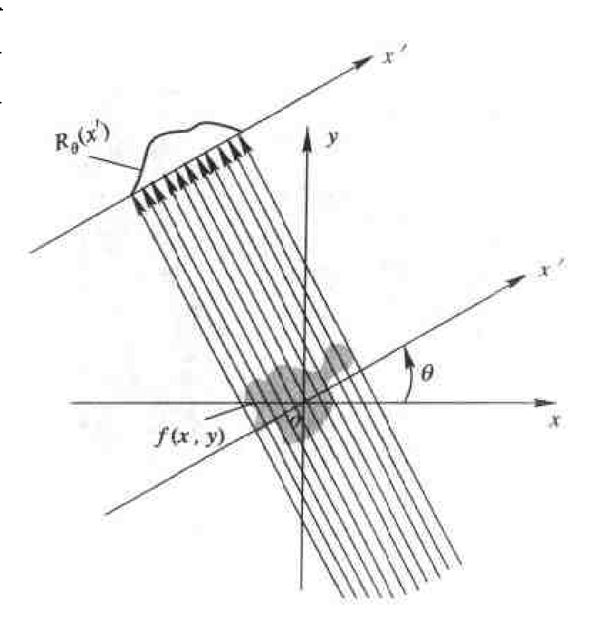



图 4.17 平行投影束重构图像原理

另一个经常使用的几何结构是扇形束几何学，这种结构由 n 个检测器和一个发送器构成。借助扇形束投影集合来构成平行束投影的方法很多，读者可以参考有关书籍。

iradon 函数使用过滤反向投影算法计算逆 Radon 变换，这个算法根据 R 列的投影构造近似的图像 I，重新构造中使用的投影越多(theta 长度越长)，重新构造的图像 IR 就越逼近于原始图像 I。向量 theta 必须包含以固定增长角度 $\Delta\theta$ 单调递增的角度值。知道标量 $\Delta\theta$ 以后，可以将其传递给 iradon 函数以代替 theta 数组(语句中表示为 Dtheta)。例如：

```
IR = iradon(R,Dtheta);
```

### 4.3.4 Radon 变换和反变换的应用实例

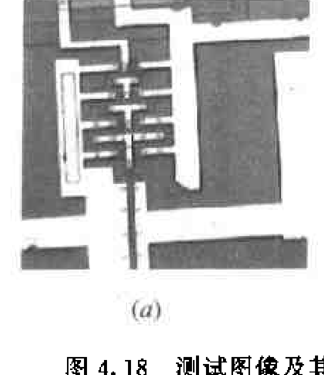
**例 4.8**：使用 Radon 变换检测图 4.18(*a*)所示图像中的直线。

检测步骤如下：

(1) 使用 edge 函数计算图像的二进制边界。

```
I = imread('ic.tif');
BW = edge(I);
subplot(1,2,1),imshow(I)
subplot(1,2,2),imshow(BW)
```

结果如图 4.18(*b*)所示。





(*a*) (*b*)

图 4.18 测试图像及其二进制边界计算结果
(*a*) 原图像；(*b*) 显示二进制边界的图像

(2) 计算边界图像的 Radon 变换。

```
theta = 0:179;
[R,xp] = radon(BW,theta);
figure,imagesc(theta,xp,R);
```

结果如图 4.19 所示。

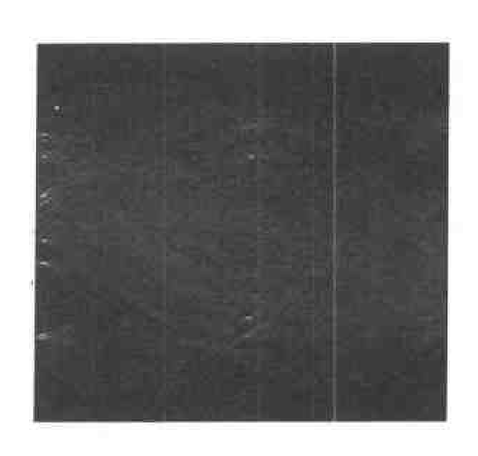


图 4.19 二进制边界的 Radon 变换结果

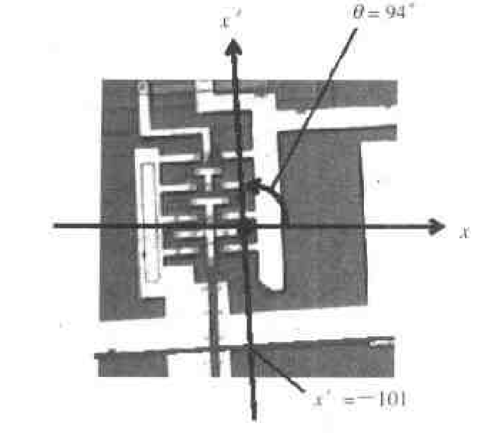



图 4.20 Radon 变换与原始图像的对应关系

(3) 找到 Radon 变换矩阵最强峰值出现的位置，这些尖峰的位置相对于原始图像中的直线位置。在本例中，R 的最强峰值相对于 $\theta=94°$，$x'=-101$。位于该位置且与该角度正交的直线如图 4.20 所示，在原始图像中用灰线着重描出，Radon 变换的几何结构用黑线标出。

⚠ **注意**：

平行于灰线的其他直线在 $\theta=94°$处的变换也会出现峰值，与这条直线正交的直线将在

$\theta=4°$处出现峰值。

△

**例 4.9**：使用 radon 和 iradon 实现采样图像的投影构造以及图像重建。

本例中的测试图像(如图 4.21 所示)是由图像处理工具箱函数 phantom 创建的，该图像说明了许多人脑具备的特征，例如，外部明亮的椭圆形外壳类似于头骨，内部许多的椭圆类似于脑瘤。

```
P = phantom(256);
imshow(P)
```

进行 Radon 变换的第一步是计算三个不同的 theta 集合：R1 采用 18 个投影，R2 采用 36 个投影，R3 采用 90 个投影：

```
theta1 = 0:10:170; [R1,xp] = radon(P,theta1);
theta2 = 0:5:175; [R2,xp] = radon(P,theta2);
theta3 = 0:2:178; [R3,xp] = radon(P,theta3);
```

使用 90 个投影(R3)进行图像的 Radon 变换：

```
figure,imagesc(theta3,xp,R3); colormap(hot); colorbar
```

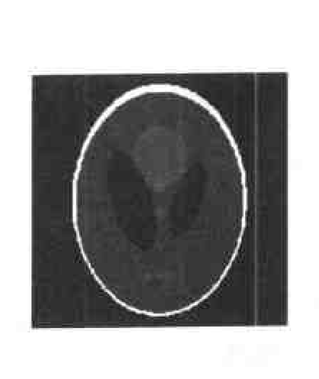


图 4.21 进行 Radon 变换及反变换测试的原始图像

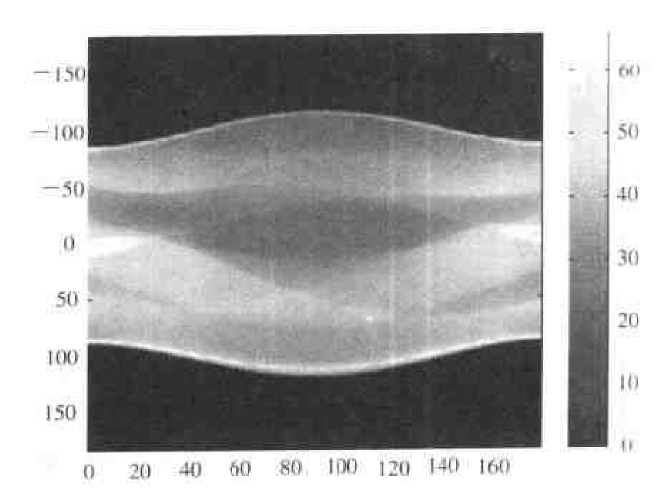


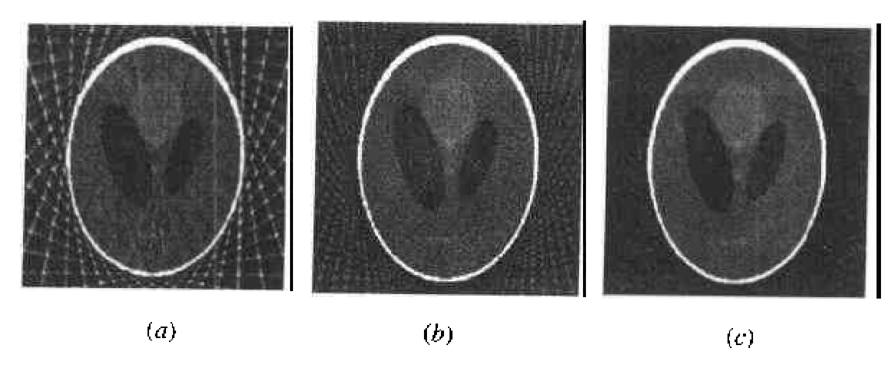

图 4.22 使用 90 个投影的 Radon 变换

观察图 4.22 可以得到输入图像的一些特征。例如，Radon 变换的第一列相对于 0°处(垂直方向的积分)的投影，最靠近中心的列相对于 90°处的投影。90°处的投影轮廓要宽于 0°处的投影，这是因为图像中最外部的椭圆具有较大的垂直半轴。

图 4.23 比较了根据以上图像利用 R1、R2 和 R3 进行 Radon 反变换后得到的重建图像。

```
I1 = iradon(R1,10);
I2 = iradon(R2,5);
I3 = iradon(R3,2);
imshow(I1)
figure,imshow(I2)
figure,imshow(I3)
```

从图 4.23 可以看出：图像 I1 重建准确率最低，这是因为它使用的投影数目最少；图像 I2 使用了 36 个投影，重建质量要好一些，但是测试图像下部的三个小椭圆仍然不能清



(a) (b) (c)

图 4.23 使用不同的投影数目重建的图像比较

(a) 使用 18 个投影；(b) 使用 36 个投影；(c) 使用 90 个投影

晰分辨；图像 I3 是使用 90 个投影重建的，结果最接近原始图像。当投影数目相对较少时，重建可能包含一些来自黑投影的人为痕迹。

# 4.4 其他图像变换技术

## 4.4.1 线性变换与基函数

设 $x$ 是 $N\times1$ 的向量，$\boldsymbol{T}$ 是一个 $N\times N$ 的矩阵，则线性变换定义为

$$y=\boldsymbol{T}x$$

或定义为

$$y_i=\sum_{j=0}^{N-1}t_{i,j}x_j \tag{4.16}$$

矩阵 $\boldsymbol{T}$ 通常称为变换的核矩阵。由于变换结果 $y$ 是由输入元素的一阶和构成的，所以称以上变换为线性变换。例如，平面坐标系中的向量旋转变换：

$$\begin{bmatrix}y_1\\y_2\end{bmatrix}=\begin{bmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} \tag{4.17}$$

其中，$\theta$ 为旋转角度。

对于一定的 $N$，$\boldsymbol{T}$ 的个数是没有限制的，但是通常只有某些具有特殊性值的 $\boldsymbol{T}$ 才具有实用价值，酉矩阵就是其中的一种。如果 $\boldsymbol{T}$ 是酉矩阵，那么下式成立：

$$\boldsymbol{T}^{-1}=(\boldsymbol{T}^*)' \quad 且\ \boldsymbol{I}=(\boldsymbol{T}^*)'\boldsymbol{T}=\boldsymbol{T}(\boldsymbol{T}^*)' \tag{4.18}$$

也就是说，酉矩阵的共轭转秩等于其逆矩阵。易知，实数酉矩阵是正交矩阵。一维 DFT 就是酉矩阵的一个例子。

如果 $\boldsymbol{T}$ 是非奇异的(通常这个条件都是成立的)，其逆矩阵存在，那么有

$$x=\boldsymbol{T}^{-1}y$$

在二维情况下，将一个 $N\times N$ 的矩阵 $f(x,y)$变换为另一个 $N\times N$ 的矩阵 $g(u,v)$的线性变换一般形式为

$$g(u,v)=\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}a(x,y,u,v)f(x,y)\quad 0\leqslant u,\ v\leqslant N-1 \tag{4.19}$$

其中，$\alpha(x,y,u,v)$是变换的核函数，该矩阵每行有 $N$ 个块，共有 $N$ 行，每个块是一个 $N\times N$ 的矩阵。块地址由 $u$、$v$ 确定，块内地址由 $x$、$y$ 确定。如果核函数能被分解成行方向的分量函数和列方向上的分量函数的乘积，即

$$\alpha(x,y,u,v) = \alpha_r(x,u)\alpha_c(y,v) \tag{4.20}$$

此时的变换就称为可分离变换，这就意味着这个变换可以由两个步骤实现：首先对行向量进行计算，然后对列向量进行计算(或者相反)，即

$$g(u,v) = \sum_{x=0}^{N-1}\Big[\sum_{y=0}^{N-1}\alpha_c(y,v)f(x,y)\Big]\alpha_r(x,u) \tag{4.21}$$

特别地，当行分量函数与列分量函数相同时，称该变换为对称变换。二维 DFT 就是一个可分离的对称酉矩阵。酉矩阵的各行构成了 $N$ 维空间的一组基向量。通常基向量都取自同一种形式的基函数。例如，傅立叶变换就是使用复指数作为基函数的原型，各个基函数之间只是频率不同。根据基函数形式的不同，可以将图像变换矩阵分为正弦型变换、方波型变换等。基函数还可以通过分析核矩阵的特征向量得出，相应的变换称为基于特征向量的变换。

在标准正交基下(单位长度的正交向量组)，空间中的任何向量都可表示成基向量的加权和。二维反变换可以看作是对一组基图像求加权和，从而重构原图像，变换矩阵中的每个元素就是其对应基图像在求和时的权值。显然，基图像也是无限多的，但相对于某种变化而言，只有一组特定的基图像集合具有重要意义。

### 4.4.2 正弦型变换

除了前面介绍的离散傅立叶变换和离散余弦变换之外，以正弦型函数为基函数的变换还包括正弦变换、哈里特变换等。

离散正弦变换(DST)对的定义如下：

$$G_s(m,n) = \frac{2}{N+1}\sum_{i=0}^{N-1}\sum_{k=0}^{N-1}g(i,k)\sin\Big[\frac{\pi(i+1)(m+1)}{N+1}\Big]\sin\Big[\frac{\pi(k+1)(n+1)}{N-1}\Big] \tag{4.22}$$

$$g(i,k) = \frac{2}{N+1}\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}G_s(m,n)\sin\Big[\frac{\pi(i+1)(m+1)}{N+1}\Big]\sin\Big[\frac{\pi(k+1)(n+1)}{N+1}\Big] \tag{4.23}$$

其中，核矩车的元素为

$$T_{i,k} = \sqrt{\frac{2}{N+1}}\sin\Big[\frac{\pi(i+1)(k+1)}{N+1}\Big]$$

DST 具有快速算法，可以用来对图像进行变换编码压缩。

哈里特变换(Hartley 变换)实际上是 DFT 的另一种计算方法，由于它避免了复数计算，从而可以大大地减少计算量。哈里特变换对定义如下：

$$G(m,n) = \frac{1}{N}\sum_{i=0}^{N-1}\sum_{k=0}^{N-1}g(i,k)\cos\Big[\frac{2\pi}{N}(im+kn)\Big] \tag{4.24}$$

$$g(i,k) = \frac{1}{N}\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}G(m,n)\cos\Big[\frac{2\pi}{N}(im+kn)\Big] \tag{4.25}$$

### 4.4.3 方波型变换

方波型变换的基函数是方波的各种变形，这种变换的计算速度都比较快，这是因为变形后的方波的乘法操作一般都非常简单。方波型变换主要有：哈达玛(Hardamard)变换、沃尔什(Walsh)变换、斜(Slant)变换和哈尔(Harr)变换等，这里简要介绍一下哈达玛变换。

哈达玛变换是对称的、可分离的酉变换，其核矩阵中只有+1和−1两种元素。哈达玛变换的维数 $N=2^n$，其中 $n$ 是整数，其核矩阵由如下方式构造：

$$\frac{1}{\sqrt{2}}H_2 = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{4.26}$$

$$\frac{1}{\sqrt{N}}H_N = \frac{1}{\sqrt{N}}\begin{bmatrix} H_{N/2} & H_{N/2} \\ H_{N/2} & -H_{N/2} \end{bmatrix} \tag{4.27}$$

例如，计算可知，对于 $N=4$，哈达玛核心矩阵为

$$H_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

### 4.4.4 基于特征向量的变换

对于 $N\times N$ 的矩阵 $\boldsymbol{A}$，若：

$$|\boldsymbol{A} - \lambda_k \boldsymbol{I}| = 0 \tag{4.28}$$

则称标量 $\lambda_k$ 为矩阵的特征值，而满足下式的向量 $\boldsymbol{v}_k$ 称为特征向量：

$$\boldsymbol{A}\boldsymbol{v}_k = \lambda_k \boldsymbol{v}_k \tag{4.29}$$

基于特征向量的变换就是通过分析矩阵的特征向量得到变换矩阵的变换方法。常用的基于特征向量的变换方法有离散K-L变换和SVD两种。离散K-L变换也叫作Hotelling变换，是以图形统计特性为基础的一种变换。离散K-L变换在模式识别中获得了广泛的应用，这里我们简单介绍一下这种变化的方法。

K-L变换的目的是寻求 $n\times n$ 的正交矩阵 $\boldsymbol{A}$，使得 $\boldsymbol{A}$ 对 $\boldsymbol{X}$ 的变换结果 $\boldsymbol{Y}$ 的协方差矩阵：

$$\Sigma_x = E\{(\boldsymbol{X} - \boldsymbol{Y})(\boldsymbol{X} - \boldsymbol{Y})^T\}$$

为对角阵。变换步骤如下：

(1) 求矩阵的特征值。

(2) 根据矩阵的特征值求特征向量。

(3) 将特征向量单位化。

(4) 对图像使用由单位化特征向量组成的变换矩阵进行变换。

K-L变换常常用来进行数据压缩。为了达到压缩的目的，通常仅保留变换后的向量 $\boldsymbol{Y}$ 的 $m$ 个分量。由证明可得，当保留 $\boldsymbol{Y}$ 分量的子集 $\overline{\boldsymbol{X}}=\{y_1,y_2,\cdots,y_m\}$ 时，要使均方误差 $\Sigma_x=E\{(\boldsymbol{X}-\overline{\boldsymbol{X}})(\boldsymbol{X}-\overline{\boldsymbol{X}})^T\}$ 最小，变换矩阵 $\boldsymbol{A}$ 必须是由 $\boldsymbol{X}$ 的协方差矩阵的特征向量构成的。因此后续步骤如下：

(5) 略去 $\Sigma_x$ 最小的 $n^2-m$ 个特征值 $\lambda$。

(6) 用所有略去分量 $y_i$ 的平均值代替每一个略去的 $y_i$。

K-L 变换的优点在于：它能够完全去除原信号中的相关性，因而具有非常重要的理论意义。其缺点在于：由于 K-L 变换的基函数取决于待变换图像的协方差矩阵，所以基函数的形式是不定的，而且计算特征值和特征向量的工作量也很大。

## 【习 题】

1. 利用 MATLAB 实现图 4.4 所示三幅图像的傅立叶变换。

2. 比较保留 20 个 DCT 变换系数重构的图像与原始图像的差别。

3. 证明：当保留 $\boldsymbol{Y}$ 分量的子集 $\overline{\boldsymbol{X}}=\{y_1,y_2,\cdots,y_m\}$ 时，要使均方误差 $\Sigma_x=E\{(\boldsymbol{X}-\overline{\boldsymbol{X}})(\boldsymbol{X}-\overline{\boldsymbol{X}})^T\}$ 最小，变换矩阵 $\boldsymbol{A}$ 必须是由 $\boldsymbol{X}$ 的协方差矩阵的特征向量构成的。

# 第五章 滤波和滤波器设计

**本章要点：**

★ 线性系统理论

★ 经典数字滤波方法

★ 魏纳滤波器及其设计方法

★ MATLAB 线性滤波器设计

## 5.1　线性系统理论

### 5.1.1　线性系统理论

系统是指任何一个接收输入并产生相应输出的实体，输入和输出可以是一维、二维和更高维数的。一般我们只关心输入和输出之间的关系，而不考虑系统的内部细节。由于线性系统理论是一门成熟的理论，通常用于描述光学系统和电路的行为，它为采样、滤波和空间分辨率的研究提供了坚实的数学基础，所以本书仅讨论有关线性系统的性质。

为了方便起见，我们首先讨论一维线性系统，然后将其推广到二维线性系统中去。对于某个特定的系统，如果输入 $x_1(t)$产生的输出为 $y_1(t)$，而另一输入 $x_2(t)$产生的输出为 $y_2(t)$，即：

$$x_1(t) \rightarrow y_1(t),\ x_2(t) \rightarrow y_2(t) \tag{5.1}$$

且满足：

$$\alpha x_1(t) + \beta x_2(t) \rightarrow \alpha y_1(t) + \beta y_2(t) \tag{5.2}$$

那么称该系统为线性系统，也就是说，线性系统的两个输入信号的线性组合所产生的输出等于单个输入所分别产生的输出的线性组合。这样，只要找到输入 $x(t)$和输出 $y(t)$的关系，就可以使用简单的信号对系统性能进行分析，这也是线性系统分析中最关键的地方。可以使用下面的方法得到输入和输出信号的关系。可以证明，线性函数表达式（叠加积分）：

$$y(t) = \int_{-\infty}^{\infty} f(t,\tau)x(\tau)\mathrm{d}\tau \tag{5.3}$$

就足够说明任何线性系统的输入与输出之间的关系。

### 5.1.2　一维卷积

对任何线性系统，一定可以选择一个二元函数 $f(t,\tau)$，对于任何 $x(\tau)$使得公式(5.3)

满足线性系统条件(5.2)式。但是，最好能够找到一个一元函数来描述线性系统，为此，引入移不变约束，即如果

$$x(t) \rightarrow y(t)$$

那么有

$$x(t-T) \rightarrow y(t-T) \tag{5.4}$$

也就是说，如果将输入信号沿时间轴平移 $T$，那么输出信号除平移同样长度之外，其他性质不变，此时称该系统为线性移不变系统。

对于线性移不变系统，根据式(5.3)和(5.4)可知：

$$y(t-T)=\int_{-\infty}^{\infty} f(t,\tau)x(\tau-T)\mathrm{d}\tau$$

令

$$t-T=t,\ \tau-T=\tau$$

得

$$y(t)=\int_{-\infty}^{\infty} f(t+T,\tau+T)x(\tau)\mathrm{d}\tau \tag{5.5}$$

式(5.3)和(5.5)说明，函数 $f$ 对于 $T$ 必须满足：

$$f(t,\tau)=f(t+T,\tau+T) \tag{5.6}$$

这意味着当两个变量增加同样的量，即 $t$ 与 $\tau$ 的差值保持不变时，$f(t,\tau)$将保持不变。于是定义：

$$g(t-\tau)=f(t,\tau) \tag{5.7}$$

则式(5.3)变为

$$y(t)=\int_{-\infty}^{\infty} g(t-\tau)x(\tau)\mathrm{d}\tau \tag{5.8}$$

这就是卷积积分的定义。式(5.8)表明，线性移不变系统的输出可以通过输入信号与一个表征系统特性的函数卷积得到，这个表征函数叫作系统的冲激响应。可以将式(5.8)简单记为

$$y=g*x \tag{5.9}$$

其中，用 * 表示两个函数的卷积。图 5.1 给出了图 5.1($a$)和图 5.1($b$)两个函数进行卷积运算的全部过程。曲线 $y(t)$上的一点可以根据以下方法得到：首先将函数 $g$ 关于原点反折并移动距离 $t$，然后计算 $x$ 与反折平移后的 $g$ 在各点的积，并对结果进行积分，这就得到了该点在 $t$ 处的输出值。对每一个 $t$ 值进行重复计算就可以得到输出曲线。当 $t$ 发生变化时，反折函数平移就会与静止不动的输入函数重叠，这两条曲线重叠部分的面积就决定了 $y(t)$ 的值。

卷积运算有几个非常实用的性质。首先，卷积具有交换性，即：

$$f*g=g*f \tag{5.10}$$

其次，卷积满足分配律：

$$f*(g+h)=f*g+f*h \tag{5.11}$$

卷积还满足结合律：

$$f*(g*h)=(f*g)*h \tag{5.12}$$

卷积求导具有以下性质：

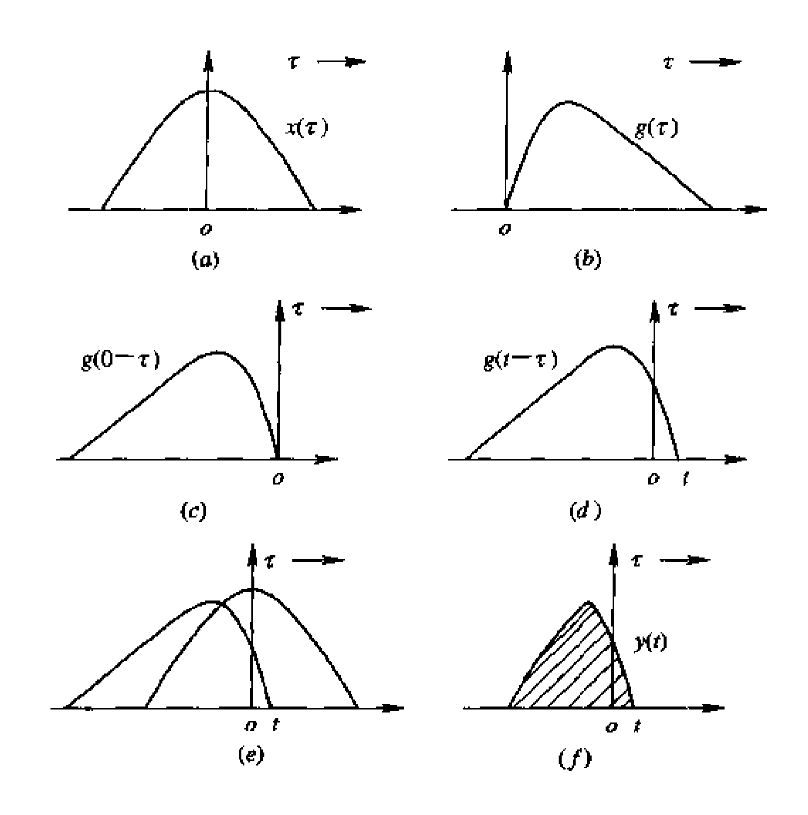



图 5.1　一维函数卷积过程

(a) 函数 x；(b) 函数 g；(c) 将函数 g 以原点进行反折；

(d) 将反折的函数 g 移动距离 τ；(e) 函数进行卷积；(f) 卷积结果

$$\frac{\partial}{\partial t}[f * g] = f' * g = f * g' \tag{5.13}$$

式(5.8)的离散形式为

$$h(i) = f(i) * g(i) = \sum_j f(j)g(i-j) \tag{5.14}$$

其中 $i=0, 1, \cdots, m$；$j=0, 1, \cdots, n$。假设 $f(i)$是一个周期为 $N$ 的无限长周期序列的一部分，一般 $f(i)$的长度都小于 $N$，那么必须对其进行补零使之长度为 $N$。对 $g(i)$和 $h(i)$同样进行补零使之周期为 $N$。令 $\boldsymbol{G}$ 是一个矩阵，其第一行是由补零后的 $g(i)$序列逆序存放，并在此基础上向右循环移动一位得到的，该矩阵的其余列都是在上一列的基础上向右循环移动一位而得到的，于是得到式(5.15)的矩阵形式如下：

$$\boldsymbol{h} = \boldsymbol{G} \cdot \boldsymbol{f} = \begin{bmatrix} g(1) & g(N) & \cdots & g(2) \\ g(2) & g(1) & \cdots & g(3) \\ \cdots & \cdots & \cdots & \cdots \\ g(N) & g(N-1) & \cdots & g(1) \end{bmatrix} \begin{bmatrix} f(1) \\ f(2) \\ \cdots \\ f(N) \end{bmatrix} \tag{5.15}$$

虽然离散函数卷积与连续函数卷积具有不同的形式，但是二者的性质是一致的，因而在原理部分常用连续函数卷积来分析，实际中都是采用离散函数卷积的形式。

### 5.1.3　二维卷积

下面将以上讨论推广到二维空间中。用 $x$ 和 $y$ 表示两个独立的自变量，二维卷积的表

达式如下：

$$h(x,y) = f * g = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(u,v)g(x-u,y-v)\mathrm{d}u\mathrm{d}v \tag{5.16}$$

二维卷积的基本原理与一维卷积基本相同，但要注意的是：二维函数的反折平移就是将函数绕原点旋转180°，然后将原点移动到$(x,y)$处。

式(5.16)的离散形式如下：

$$H(i,j) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} F(m,n)G(i-m,j-n) \tag{5.17}$$

其中$m=0,1,\cdots,M-1$；$n=0,1,\cdots,N-1$。一般将$\boldsymbol{G}$称为卷积核。

如果$\boldsymbol{F}$的大小为$m_1\times n_1$，$\boldsymbol{G}$的大小为$m_2\times n_2$，那么通过零填充将其扩展为$M\times N$。首先将$\boldsymbol{F}$的第一行转置，使之成为$\boldsymbol{F}$的最上面$M$个元素，然后将其他行转置依次放在其下。之后将矩阵$\boldsymbol{G}$的每一行都采用循环右移一位的方法生成一个$M\times M$的循环矩阵$\boldsymbol{G}_i$，其中$i=0,1,\cdots,b$，表示$\boldsymbol{G}$的行数。将$N$个这样的矩阵构造一个矩阵$\boldsymbol{G}_b$：

$$\boldsymbol{G}_b = \begin{bmatrix} [G_1] & [G_N] & \cdots & [G_2] \\ [G_2] & [G_1] & \cdots & [G_3] \\ \vdots & \vdots & \vdots & \vdots \\ [G_N] & [G_{N-1}] & \cdots & [G_1] \end{bmatrix} \tag{5.18}$$

于是式(5.18)可以写为

$$\boldsymbol{h} = \boldsymbol{G}_b \cdot \boldsymbol{f} \tag{5.19}$$

卷积常用来实现对信号或图像进行线性运算(具体方法参见第七章)，在图像滤波、去卷积、去噪声和增强等方面用处较大。例如，图5.2给出了一种用来增强图像对象边缘的滤波技术。边缘函数$f(x)$缓慢地从低变到高，脉冲响应$g(x)$是一个有着负旁瓣的正尖峰函数。随着卷积过程的进行，$g(x)$将从左移到右，旁瓣和主尖峰依次与边缘相遇，输出的增强效果$h(x)$如图5.2所示。

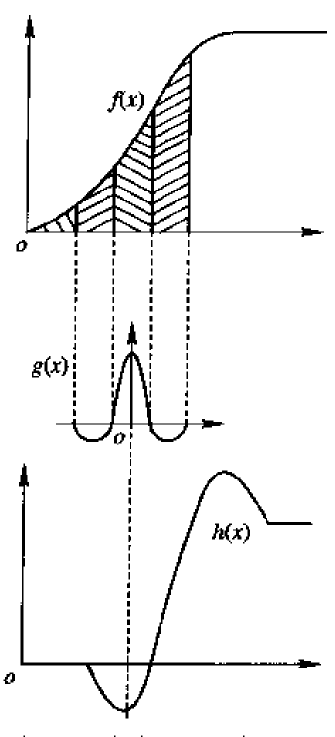


图5.2　边缘增强示意图

### 5.1.4　卷积定理和相关性定理

卷积定理是线性系统分析中最常用、最重要的定理之一，该定理指出了傅立叶变换的一个重要性质：时域中的卷积相当于频域中的相乘，即：

$$\mathscr{F}\{f(t) * g(t)\} = F(s)G(s) \tag{5.20}$$

且

$$\mathscr{F}^{-1}\{F(s)G(s)\} = f(t) * g(t) \tag{5.21}$$

由式(5.20)可知，时域中复杂的卷积运算可以在频域中通过简单的乘法来实现：首先对两个函数进行傅立叶变换，然后求它们的乘积，最后求出乘积的傅立叶逆变换就得到了两个函数的卷积。如果需要利用卷积进行图像的滤波或其他运算时，使用卷积定理可以大大减少运算的复杂程度。用卷积定理还可以证明：任何函数和冲激函数卷积后将保持不

变，由此证明冲激函数的傅立叶变换是单位 1。

相关性定理也是线性系统理论中比较重要的一个定理，该定理描述了函数自变量尺度变换对其傅立叶变换的影响：减小自变量的尺度将会展宽函数的傅立叶变换频谱，而增大自变量的尺度则会压缩函数的傅立叶变换频谱。这是因为，根据傅立叶变换的性质可以推出以下式子：

$$\mathscr{F}\{f(at)\} = \frac{1}{|a|}F\left(\frac{s}{a}\right) \tag{5.22}$$

虽然一般并不会直接利用相关性定理进行图像处理，但是可以利用这个定理对图像处理中出现的一些问题作出分析和解释。

### 5.1.5　滤波与滤波器设计

滤波是信号处理的一种最基本而又极为重要的技术，利用滤波技术可从复杂的信号中提取出所需要的信号，抑制不需要的信号。所谓滤波器就是一种选频器件或结构，它对某一频率的信号给予很小的衰减，使这部分信号能顺利通过，而对其他不需要的频率信号则进行大幅度衰减，尽可能阻止这些信号通过。在图像处理中，滤波常常用来修改或增强图像，以提高图像的信息量。例如，可以对一幅图像进行滤波，从而强调或消除其主要特征。

假设线性系统：

$$y(n) = x(n) * h(n)$$

若 $x(n)$、$y(n)$的傅立叶变换存在，则输入、输出的频域关系是

$$Y(e^{j\omega}) = X(e^{j\omega}) \cdot H(e^{j\omega}) \tag{5.23}$$

假定$|X(e^{j\omega})|$、$|H(e^{j\omega})|$分别如图 5.3(*a*)、5.3(*b*)所示，那么由式(5.23)可知，$|Y(e^{j\omega})|$将如图 5.3(c)所示。

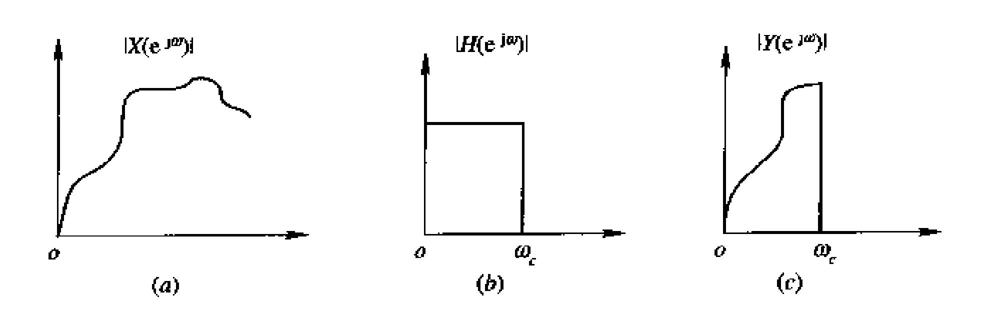



图 5.3　滤波原理图

(*a*) $|X(e^{j\omega})|$；(*b*) $|H(e^{j\omega})|$；(*c*) $|Y(e^{j\omega})|$

图 5.3 中的 $\omega_c$ 称为截止频率，它是信号通带与阻带的分界频率(以很小的衰减通过滤波器的频率范围称为滤波器的通带；被滤波器阻止通过的频率范围称为阻带)。这样，$x(n)$通过系统 $h(n)$的结果是使输出 $y(n)$中不再含有$|\omega| \geq \omega_c$的频率成分，而$|\omega| < \omega_c$的成分则毫不失真。因此，通过设计通频带，不同的$|H(e^{j\omega})|$就可以获得不同的滤波效果。

若滤波器的输入、输出都是离散时间信号，那么该滤波器的冲激响应也必然是离散的，这样的滤波器称为数字滤波器。按照滤波器的时域特性可以将数字滤波器分为无限冲激响应滤波器(IIR 滤波器)和有限冲激响应滤波器(FIR 滤波器)两种。IIR 滤波器由于不具

备FIR所固有的稳定性和设计实现的简单性，所以一般不使用这种滤波器。MATLAB所有滤波器设计函数都将返回一个FIR滤波器。FIR滤波器是一个对单点或脉冲具有有限响应范围的滤波器，由于它具有描述方便(可以用系数矩阵表示)、可以有效防止图像失真(相位呈线性)、设计可靠、容易实现、效果稳定等特征，所以成为MATLAB环境下最理想的滤波工具。

从应用理论的角度来看，数字滤波器可分为两大类：经典滤波器和现代滤波器。经典滤波器是假定输入信号中的有用频率成分和希望去除的频率成分各占有不同的频带。经典滤波器又可以分为四种，即低通(LP)滤波器、高通(HP)滤波器、带通(BP)滤波器和带阻(BS)滤波器。低通滤波器使截止频率以下的所有信号通过，截止频率以上的信号则给予很大的衰减，阻止其通过；高通滤波器使截止频率以上的信号通过，阻止截止频率以下的信号通过；带通滤波器使某一频带内的信号通过，而对于这个频带范围以外的信号则给予很大的衰减；抑制某一频带内的信号，同时让这一频带以外的其他信号通过，这样的滤波器称为带阻滤波器。带通滤波器与带阻滤波器中有两个截止频率，分别称为上截止频率和下截止频率。

如果信号和噪声的频谱相互重叠，那么就需要使用现代滤波器。现代滤波器主要研究如何从含有噪声的数据记录(又称时间序列)中估计出信号的某些特征或信号本身。通常估计信号的信噪比将会高于原始信号。现代滤波器把信号和噪声都视为随机信号，利用它们的统计特征(如，自相关函数、功率谱等)导出一套最佳的估值算法，然后予以实现。魏纳(Wiener)滤波是一种最基本而常用的现代滤波方法，这种滤波器以最小均方误差作为最优标准，通过分析输入随机信号和噪声的自相关性，利用傅立叶变换得到原始信号的最优估计。

## 5.2　经典数字滤波方法

### 5.2.1　低通滤波器

通常信号或图像的大部分能量都集中在其频谱的中、低频段，而在高频部分，图像中有用的信息常被噪声淹没，因此，可以用一个降低高频成分的滤波器来达到一种图像平滑效果。最简单的低通滤波器有三种：矩形滤波器、三角形滤波器、高频截止滤波器。

矩形滤波器将输入信号与矩形脉冲做卷积，从而实现简单的信号局部平均。矩形滤波器的传递函数及其傅立叶变换表达式分别如下：

$$f(x) = \Pi\left(\frac{x}{2a}\right) \tag{5.24}$$

$$F(s) = 2a\frac{\sin(2\pi as)}{2\pi as} \tag{5.25}$$

图5.4给出了两个宽度不同的矩形脉冲信号及其傅立叶变换曲线，其中$a<b$。根据相关性定理，传递函数的宽度与冲激响应的宽度成反比，如果矩形滤波器的宽度超过图像的两个像素大小，那么图像中的细微结构就很有可能出现极性反转的现象，这是因为此时矩形滤波器的傅立叶变换变化十分陡峭，使像素的灰度发生变化。

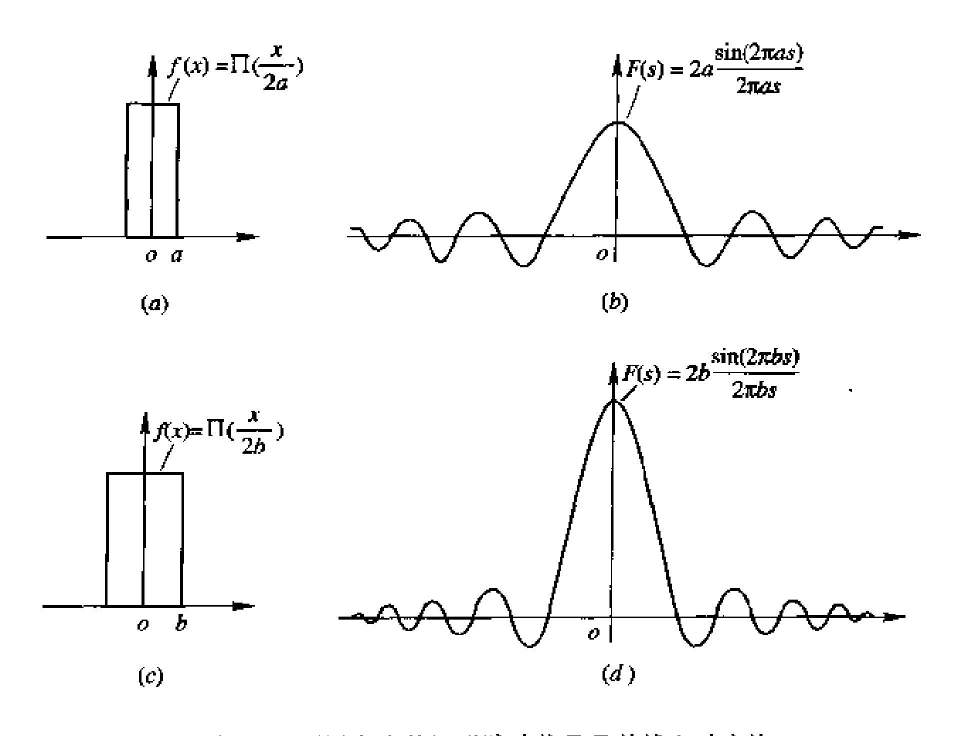



图 5.4　不同宽度的矩形脉冲信号及其傅立叶变换

(a) 窄矩形脉冲信号；(b) (a)信号的傅立叶变换；

(c) 宽矩形脉冲信号；(d) (c)信号的傅立叶变换

三角形滤波器以三角形脉冲 $\Lambda(t)$ 作为滤波器的冲激响应，在二维情况下表现为金字塔形。三角形脉冲的频谱具有 $[\sin(x)/x]^2$ 的形式，其数值始终保持为正数，而且随着频率的提高，其信号衰减速度远远大于矩形滤波器。图 5.5 给出了三角形脉冲信号及其傅立叶变换曲线。由于三角形滤波器的宽度对其傅立叶变换的影响并不是很大，所以其宽度限制没有矩形滤波器那么严格，可以在较大宽度内安全使用而不必担心像素极性反转。

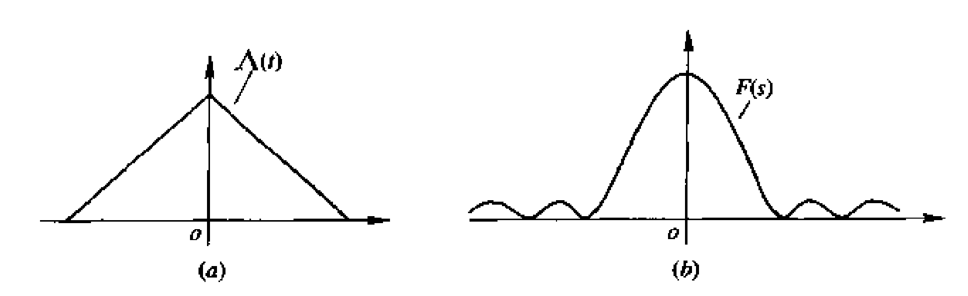



图 5.5　三角形脉冲信号及其傅立叶变换

(a) 三角形脉冲信号；(b) 三角形脉冲信号的傅立叶变换

高频截止滤波器是一种较为粗略的低通滤波方法，这种滤波方法首先计算信号或图像的傅立叶变换，然后将傅立叶变换幅值谱的高频部分强行设置为零，再求出傅立叶反变换得到滤波后的图像。有时采用这种方法会在图像的尖峰或边界附近产生振铃效应，因而其用途有限。除了以上介绍的三种简单滤波器，还有几种在图像处理中经常用到的较为复杂的滤波器，例如，高斯低通滤波器、指数低通滤波器和巴特沃斯低通滤波器等，这些滤波器的区别在于其传递函数不同。例如，高斯低通滤波器使用高斯函数作为滤波器的传递函数。

$$f(t) = e^{-\pi t^2} \tag{5.26}$$

可以证明，高斯函数的傅立叶变换仍然是高斯函数，即高斯函数的傅立叶变换为

$$F(s) = e^{-\pi s^2} \tag{5.27}$$

因此，高斯函数可以构成一个具有平滑性能的低通滤波器。上述几种复杂滤波器的工作原理及使用方法将在以后的章节中陆续向大家作以详细介绍。

### 5.2.2　带通和带阻滤波器

首先介绍理想带通滤波器。假设我们希望利用卷积实现一个仅允许位于频率 $f_1$ 和 $f_2$ 之间信号通过的滤波器，那么这个滤波器的传递函数形式为

$$G(s) = \begin{cases} 1 & f_1 \leqslant |s| \leqslant f_2 \\ 0 & \text{其他} \end{cases} \tag{5.28}$$

图 5.6(*a*)给出了带通滤波器传递函数的形状，事实上这种滤波器是一种理想带通滤波器，在物理上是无法实现的。由于 $G(s)$是由一对矩形脉冲组成的，所以可以将它视为矩形脉冲与冲激对做卷积构成的。令：

$$s_0 = \frac{1}{2}(f_1 + f_2) \quad \Delta s = f_2 - f_1$$

则理想带通滤波器的传递函数为

$$G(s) = \Pi\left(\frac{s}{\Delta s}\right) * [\delta(s - s_0) + \delta(s + s_0)] \tag{5.29}$$

那么理想带通滤波器的冲激响应(即 $G(s)$的傅立叶反变换)为

$$g(t) = 2\Delta s \frac{\sin(\pi\Delta st)}{\pi\Delta st} \cos(2\pi s_0 t) \tag{5.30}$$

理想带通滤波器的冲激响应曲线是一个被频率为 $\Delta s/2$ 的函数曲线 $\sin x/x$ 包围的余弦波，该余弦波的频率为 $s_0$(参见图 5.6(*b*))。

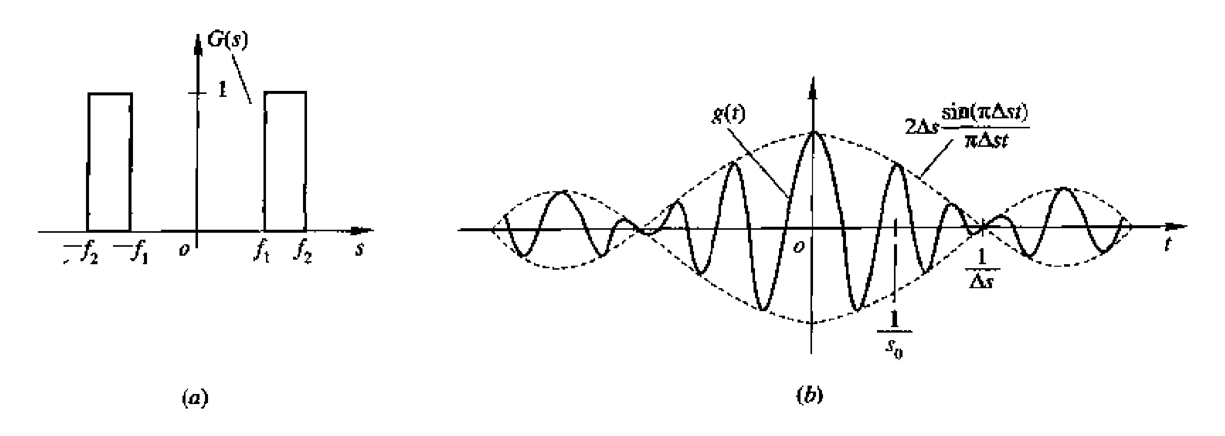



图 5.6　理想带通滤波器的传递函数及冲激响应曲线
(*a*) 理想带通滤波器的传递函数；(*b*) 理想带通滤波器的冲激响应

理想带阻滤波器与理想带通滤波器的传递函数正好相反，其表达式如下：

$$G(s) = \begin{cases} 0 & f_1 \leqslant |s| \leqslant f_2 \\ 1 & \text{其他} \end{cases} \tag{5.31}$$

显然，根据式(5.28)、(5.29)和(5.31)可以将该传递函数写为

$$G(s)=1-\Pi\left(\frac{s}{\Delta s}\right)*[\delta(s-s_0)+\delta(s+s_0)] \tag{5.32}$$

由此可以得出理想带阻滤波器的冲激响应为

$$g(t)=\delta(t)-2\Delta s\frac{\sin(\pi\Delta st)}{\pi\Delta st}\cos(2\pi s_0t) \tag{5.33}$$

理想带阻滤波器的传递函数和冲激响应曲线如图 5.7(*a*)、(*b*)所示。

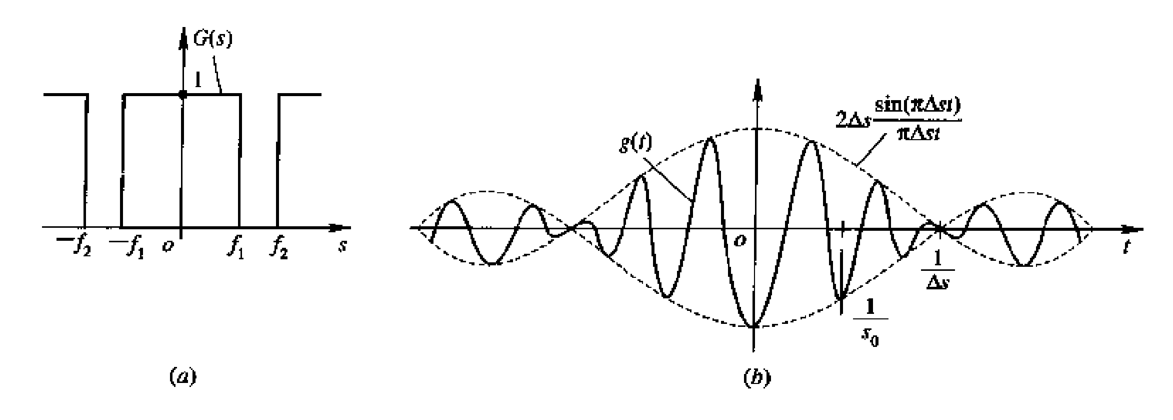



图 5.7　理想带阻滤波器的传递函数及冲激响应曲线

(*a*) 理想带阻滤波器的传递函数；(*b*) 理想带阻滤波器的冲激响应

工程上使用的可实现的带通和带阻滤波器都是选取一个非负的单峰函数 $K(s)$代替矩形脉冲与位于频率 $s_0$ 处的冲激对做卷积得到的。对于带通滤波器，其传递函数为

$$G(s)=K(s)*[\delta(s-s_0)+\delta(s+s_0)] \tag{5.34}$$

对于带阻滤波器，其传递函数为

$$G(s)=1-K(s)*[\delta(s-s_0)+\delta(s+s_0)] \tag{5.35}$$

相应的冲激响应分别为

$$g(t)=2k(t)\cos(2\pi s_0t) \tag{5.36}$$

$$g(t)=\delta(t)-2k(t)\cos(2\pi s_0t) \tag{5.37}$$

函数 $K(s)$可以是高斯函数、指数函数等，相应的滤波器分别称为高斯带通(或带阻)滤波器、指数带通(或带阻)滤波器等。例如，假设选择 $K(s)$为高斯函数，那么相应的高斯带通滤波器的传递函数为

$$G(s)=Ae^{-s^2/2a^2}*[\delta(s-s_0)+\delta(s+s_0)] \tag{5.38}$$

冲激响应为

$$g(t)=\frac{2A}{\sqrt{2\pi\sigma^2}}e^{-t^2/2\sigma^2}\cos(2\pi s_0t) \tag{5.39}$$

其中

$$\sigma=\frac{1}{2\pi a}$$

高斯带通滤波器的传递函数和冲激响应曲线分别如图 5.8(*a*)、(*b*)所示。

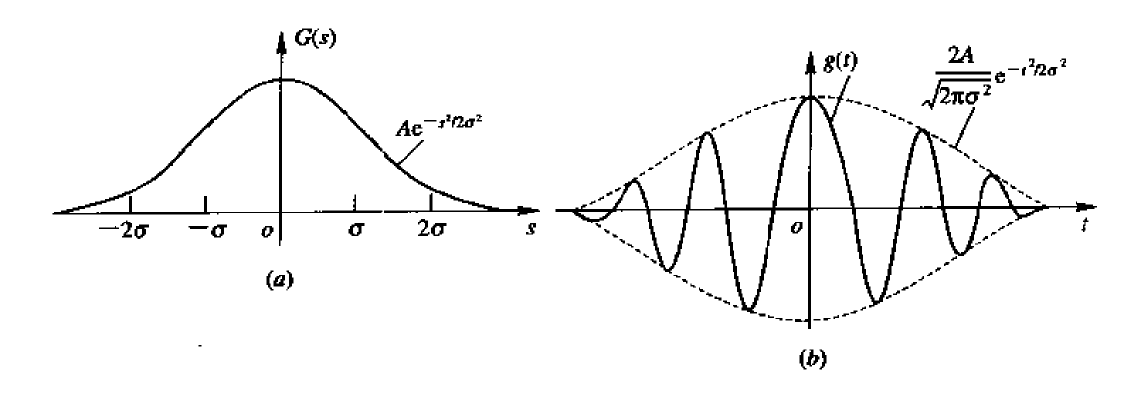



图 5.8 高斯带通滤波器传递函数及冲激响应曲线

(a) 高斯带通滤波器的传递函数；(b) 高斯带通滤波器的冲激响应

### 5.2.3 高通滤波器

高通滤波器也称为高频增强滤波器，其传递函数在 0 频率处取值为单位 1，随着频率的增长，传递函数的数值逐渐增加，当频率增加到一定数值后传递函数又取 0 值或降低某个大于 1 的增益。如果传递函数通过原点，那么称该滤波器为拉普拉斯(Laplacian)滤波器。通用高通滤波器的传递函数和冲激响应曲线如图 5.9(a)、(b)所示。

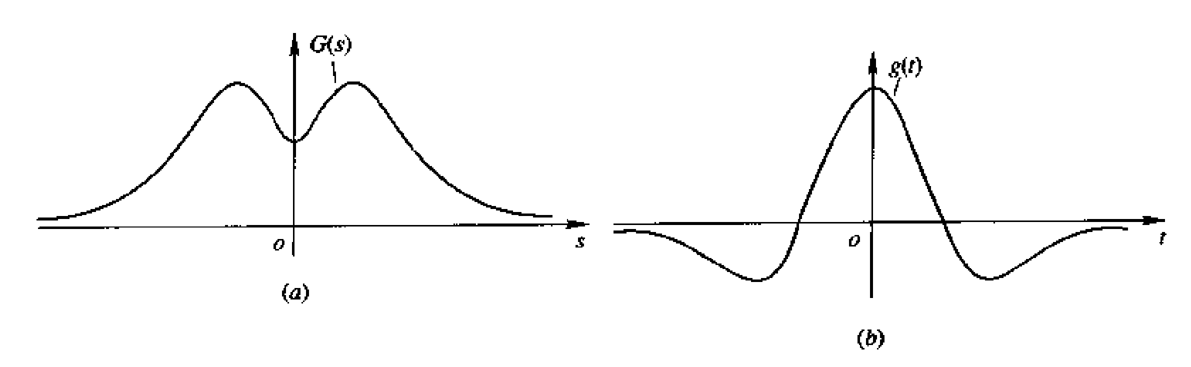



图 5.9 高通滤波器的传递函数及冲激响应曲线

(a) 高通滤波器的传递函数；(b) 高通滤波器的冲激响应

通常，假设将高通滤波器的冲激响应表示为一个窄脉冲减去一个宽脉冲，即：

$$g(t) = g_1(t) - g_2(t) \tag{5.40}$$

由于高通滤波器的传递函数在 0 频率处的取值将决定该滤波器对图像中大目标对比度的影响，所以在设计高通滤波器时要着重考虑传递函数的 0 频率取值。对式(5.40)进行傅立叶变换，并令 $s=0$，则

$$G(0) = \int_{-\infty}^{\infty} g(t)\mathrm{d}t = \int_{-\infty}^{\infty} g_1(t)\,\mathrm{d}t - \int_{-\infty}^{\infty} g_2(t)\,\mathrm{d}t = A_1 - A_2 \tag{5.41}$$

其中，$A_1$ 和 $A_2$ 分别代表两个函数分量下的面积。在设计高通滤波器时可以根据式(5.41)来确定两个脉冲函数，最好使它们的面积差等于单位 1，从而保证图像中大而平坦区域的对比度不变。

另外，还要对所需高通滤波器的传递函数的最大值做出估计，然后通过傅立叶变换分析确定构成高通滤波器的两个脉冲信号。

# 5.3　魏纳滤波器及其设计方法

## 5.3.1　随机变量

前面已经提到过，魏纳滤波器将信号和噪声都视为随机信号，在对这些随机信号进行统计分析的基础上设计出符合最优准则的滤波器。随机实际上就代表对图像知识的缺乏，在对成像的物理机理了解不够详细，甚至一无所知时就会出现这种信息缺乏，正是由于这种缺乏使我们没有办法写出噪声或输入信号的数学表达式，因而无法进行信号的定性、定量的分析。

可以按照这样的方法来处理随机变量：考虑一个由无限多个函数成员构成的样本集，假设其中的某一个成员函数(但是不知道具体是哪一个成员)在记录时出现了噪声污染，那么可以通过对样本集整体做出一些总体描述，用这些总体描述来表示有污染信号的部分情况。为了分析处理随机变量，引入期望算子：

$$\varepsilon\{x(t)\} = \int_{-\infty}^{\infty} x(t)\mathrm{d}t \tag{5.42}$$

该算子表示随机变量 $x$ 在 $t$ 时刻的样本集均值。假设已知信号 $x(t)$ 的自相关函数：

$$R_{x}(\tau) = \int x(t)x(t+\tau)\mathrm{d}t \tag{5.43}$$

那么 $x(t)$ 的功率谱

$$P_{x}(s) = \mathscr{F}\{R_{x}(\tau)\} \tag{5.44}$$

也是已知的，这就说明信号 $x(t)$ 的幅值谱是已知的，但是相位谱未知。实际上成员函数集就是由无限多个仅在相位上有区别的函数均值构成的。符合式(5.43)和(5.44)的随机信号称为遍历性随机信号，在实际应用中遇到的随机信号大部分都是这种类型的信号。

## 5.3.2　魏纳滤波原理

为了说明简单起见，这里首先介绍一维魏纳滤波器，然后再将其推广到二维的情况。假设信号 $x(t)$ 是由有用信号 $s(t)$ 和噪声信号 $n(t)$ 构成的，设计滤波器的目的就是使输出信号 $y(t)$ 尽可能地降低噪声信号 $n(t)$，同时恢复有用信号 $s(t)$。在开始设计滤波器之前，首先要建立一个最优标准，使滤波器估计所得的信号按照这个标准来说是最优的。当然，最精确的最优准则就是 $y(t)=s(t)$，但这是线性滤波器无法做到的。

定义误差信号：

$$e(t) = s(t) - y(t) \tag{5.45}$$

那么均方误差就是平均误差的度量：

$$MSE = \varepsilon\{e^{2}(t)\} = \int_{-\infty}^{\infty} e^{2}(t)\,\mathrm{d}t \tag{5.46}$$

魏纳滤波器以最小化均方误差作为最优准则。这是因为对误差进行平方运算将使得大误差的分量远远大于小误差分量，选择最小化均方误差就可以限制滤波器输出的主要误差。也可以使用其他的最优准则进行分析(例如，平均误差等)，但是这些准则将使得分析过程变得较为复杂，而且效果不是很好。

现在可以将魏纳滤波描述为：给定 $s(t)$和 $n(t)$的功率谱，选择一个均方误差最小的冲激响应 $h(t)$，使得输出 $y(t)=x(t)*h(t)$产生的均方误差最小。可以看出，均方误差实际上是冲激响应$h(t)$的函数，将$h(t)$映射为实数$MSE$，因此魏纳滤波器的设计问题可以归结为求取函数$h(t)$，使$MSE$最小。我们使用变分法来求取$h(t)$。

由式(5.46)可知：

$$MSE = \varepsilon\{[s(t)-y(t)]^2\} = \varepsilon\{s^2(t)-2s(t)y(t)+y^2(t)\} \tag{5.47}$$

根据期望算子的性质可知：

$$MSE = \varepsilon\{s^2(t)\} - 2\varepsilon\{s(t)y(t)\} + \varepsilon\{y^2(t)\} = T_1 + T_2 + T_3 \tag{5.48}$$

下面分别考虑 $T_1$、$T_2$ 和 $T_3$。将 $T_1$ 写为积分形式：

$$T_1 = \int_{-\infty}^{\infty} s^2(t)\mathrm{d}t = R_s(0) \tag{5.49}$$

从式(5.49)可以看出，$T_1$ 实际上就是已知的 $s(t)$自相关函数在 0 点处的取值，所以 $T_1$ 实际上是已知的。因为 $y(t)=x(t)*h(t)$，所以

$$T_2 = -2\varepsilon\left\{s(t)\int_{-\infty}^{\infty} h(\tau)x(t-\tau)\mathrm{d}\tau\right\} \tag{5.50}$$

根据期望算子的性质以及互相关函数的定义：

$$R_{fg}(\tau) = f(t)*g(-t) = \int f(t)g(t+\tau)\mathrm{d}\tau \tag{5.51}$$

可以将式(5.50)写为

$$T_2 = -2\int_{-\infty}^{\infty} h(\tau)R_{xs}(\tau)\mathrm{d}\tau \tag{5.52}$$

将 $T_3$ 写为两个卷积乘积的形式：

$$T_3 = \varepsilon\left\{\int_{-\infty}^{\infty} h(\tau)x(t-\tau)\mathrm{d}\tau\int_{-\infty}^{\infty} h(u)x(t-u)\mathrm{d}u\right\} \tag{5.53}$$

整理可知：

$$T_3 = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h(\tau)h(u)\varepsilon\{x(t-\tau)x(t-u)\}\,\mathrm{d}\tau\,\mathrm{d}u \tag{5.54}$$

令

$$\varepsilon\{x(t-\tau)x(t-u)\} = R_x(u-\tau)$$

故有

$$T_3 = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h(\tau)h(u)R_x(u-\tau)\,\mathrm{d}\tau\,\mathrm{d}u \tag{5.55}$$

于是可以将式(5.48)写为

$$MSE = R_s(0) - 2\int_{-\infty}^{\infty} h(\tau)R_{xs}(\tau)\mathrm{d}\tau + \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h(\tau)h(u)R_x(u-\tau)\,\mathrm{d}\tau\,\mathrm{d}u \tag{5.56}$$

从式(5.56)可以看出，$MSE$ 是函数 $h(t)$的函数，现在选择 $h_0(t)$，使 $MSE$ 取最小值。令

$$h(t) = h_0(t) + g(t)$$

则式(5.56)可以写为

$$MSE = R_s(0) - 22\int_{-\infty}^{\infty}[h_0(\tau) + g(\tau)]R_{xs}(\tau)\mathrm{d}\tau$$
$$+ \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}[h_0(\tau) + g(\tau)][h_0(u) + g(u)]R_s(u-\tau)\,\mathrm{d}\tau\,\mathrm{d}u \tag{5.57}$$

将式(5.57)展开得

$$MSE = R_s(0) - 2\int_{-\infty}^{\infty}h_0(\tau)R_{xs}(\tau)\mathrm{d}\tau + \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}h_0(\tau)h_0(u)R_s(u-\tau)\,\mathrm{d}\tau\,\mathrm{d}u$$
$$+ \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}h_0(\tau)g(u)R_s(u-\tau)\,\mathrm{d}\tau\,\mathrm{d}u + \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}g(\tau)h_0(u)R_s(u-\tau)\,\mathrm{d}\tau\,\mathrm{d}u$$
$$- 2\int_{-\infty}^{\infty}g(\tau)R_{xs}(\tau)\,\mathrm{d}\tau + \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}g(\tau)g(u)R_s(u-\tau)\,\mathrm{d}\tau\,\mathrm{d}u \tag{5.58}$$

从式(5.58)中可以看出，该式的前三项就是由函数 $h_0(t)$产生的均方误差，记这个均方误差为 $MSE_0$。另外，可以证明该式的最后一项与 $h_0(t)$无关，且保持为正数，将该项记为 $T_4$，则式(5.58)可以写为

$$MSE = MSE_0 + 2\int_{-\infty}^{\infty}g(u)\left[\int_{-\infty}^{\infty}h_0(\tau)R_x(u-\tau)\mathrm{d}\tau - R_{xs}(u)\right]\mathrm{d}u + T_4 \tag{5.59}$$

可以证明，式(5.59)中的第二项对于所有的 $u$ 取值都为 0 是 $MSE \geqslant MSE_0$ 的充要条件。进一步也可以证明下式：

$$R_{xs}(u) = \int_{-\infty}^{\infty}h_0(\tau)R_x(u-\tau)\mathrm{d}\tau \tag{5.60}$$

是 $MSE \geqslant MSE_0$ 的充要条件。我们注意到式(5.60)右边项实际上是一个卷积积分，即最优条件下($h(t)=h_0(t)$)滤波器的输出。容易证明，对于任何线性系统，输入、输出的互相关函数为

$$R_{xy}(\tau) = h(u) * R_x(u)$$

那么，根据式(5.60)可知：

$$R_{xs}(\tau) = h_0(u) * R_x(u) = R_{xy}(\tau) \tag{5.61}$$

对式(5.61)两边进行傅立叶变换得

$$P_{xs}(s) = H_0(s) * P_x(s) = P_{xy}(s) \tag{5.62}$$

则

$$H_0(s) = \frac{P_{xs}(s)}{P_x(s)} \tag{5.63}$$

这就得到了魏纳滤波器的传递函数。

### 5.3.3 魏纳滤波器设计方法

由以上的分析可知，魏纳滤波器的整个设计步骤如下：

(1) 对输入信号 $s(t)$的样本进行数字化。

(2) 求输入样本的自相关函数，从而得到 $R_x(\tau)$的一个估计值。

(3) 计算 $R_x(\tau)$的傅立叶变换，得到 $P_x(s)$。

(4) 在无噪声情况下对输入信号的一个样本进行数字化。

(5) 求信号样本与输入样本的互相关函数，从而估计出 $R_{xs}(\tau)$。

(6) 计算 $R_{xs}(\tau)$的傅立叶变换，得出 $P_{xs}(s)$。

(7) 利用式(5.63)计算魏纳滤波器的传递函数 $H_0(s)$。

如果需要使用卷积运算来实现魏纳滤波，那么可以通过计算的傅立叶反变换得到冲激响应 $h_0(t)$。这里要注意，如果无法得到无噪声信号和输入信号的样本，那么可以指定相应的自相关函数或功率谱形式进行计算。比较常用的形式就是功率谱恒定的白噪声。

图 5.10 给出了一个魏纳滤波器的示例。假设信号和噪声的功率谱如图 5.10(*a*)所示，计算得出的滤波器传递函数如图 5.10(*b*)所示，相应的均方误差曲线如图 5.10(*c*)所示。

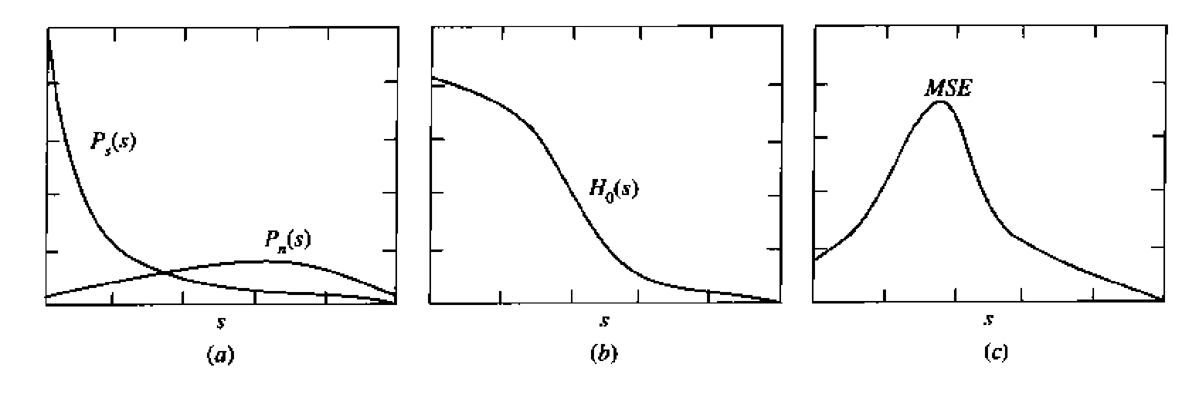



图 5.10　魏纳滤波器传递函数和均方误差曲线

(*a*) 信号和噪声的功率谱；(*b*) 滤波器传递函数；(*c*) 均方误差曲线

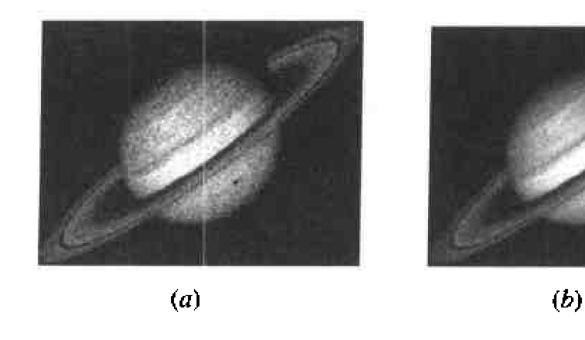
## 5.3.4　图像的魏纳滤波方法

对图像进行魏纳滤波主要是为了消除图像中存在的噪声。可以利用 MATLAB 的 wiener2 函数对一幅图像进行自适应魏纳滤波。wiener2 函数的调用格式如下：

```
J = wiener2 (I,[M N],NOISE)
```

其中，I 表示输入图像，[M N]表示卷积使用的邻域大小，缺省值为[3 3]，NOISE 是噪声强度，如果不指定此参数，那么 wiener2 函数将返回一个估计的噪声强度。以下代码将对图 5.11(*a*)所示的图像进行魏纳滤波，滤波结果如图 5.11(*b*)所示。

```
I = imread('saturn.tif');
J = imnoise(I,'gaussian',0,0.005);
[K noise] = wiener2(J,[5 5]);
subplot(1,2,1),imshow(J);
subplot(1,2,2),imshow(K);
```



(*a*)　　(*b*)

图 5.11　魏纳滤波前、后显示效果比较

(*a*) 原图像；(*b*) 滤波后

魏纳滤波估计的噪声强度(noise 的返回值)为 0.0039，与实际噪声强度有一定的差别。如果指定噪声强度为 0.005 再进行滤波，那么效果要好一些(如图 5.12 所示)。

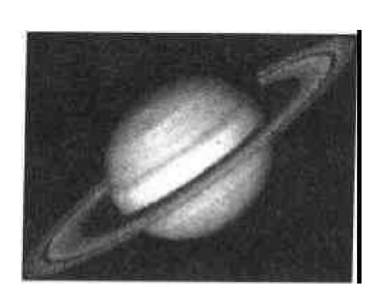


图 5.12　指定噪声强度后的滤波效果

# 5.4　MATLAB 线性滤波器设计

## 5.4.1　MATLAB 线性滤波器设计

从以上的介绍中可以看出，不论哪一种滤波方式，最关键的地方就在于滤波器的设计。为了简化计算和减少运算时间，MATLAB 的线性滤波器设计都是建立在频域中的，主要的设计方法有三种：频率变换方式、频率采样方式(根据所需频率响应创建滤波器)和窗口方式(将理想脉冲响应乘以窗口函数从而生成所需滤波器)。MATLAB 的工具箱中没有直接用来设计魏纳滤波器的函数，读者可以使用 5.3 节中介绍的方法利用傅立叶变换自行设计，此处不再赘述。这里要注意的是，如果没有安装信号处理和滤波器设计工具箱，那么在图像处理工具箱中设计滤波器将非常困难。

## 5.4.2　频率变换方式

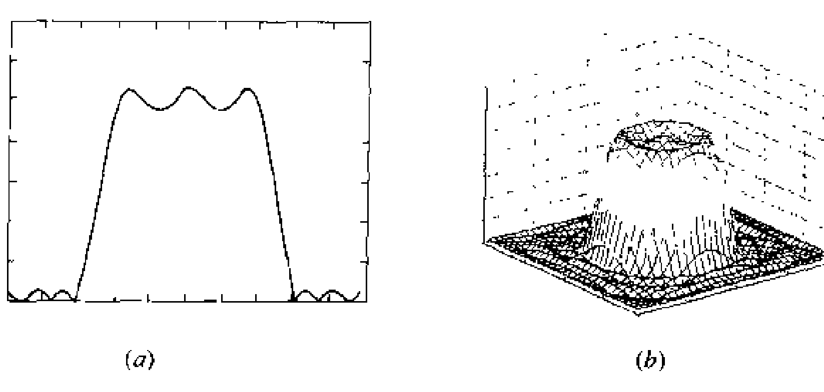
频率变换设计方法首先设计一个性能较好的一维 FIR 滤波器，然后将其变换为二维 FIR 滤波器。由于设计一个具有特殊性质的一维滤波器比二维滤波器要简单得多，所以频率变换方法通常能够产生很好的效果。函数 ftrans2 可以实现频率变换滤波器的设计。在缺省情况下该函数将生成一个几乎完全中心对称的滤波器，当然，也可以通过指定变换矩阵从而获得其他对称方式的滤波器。ftrans2 的调用格式如下：

```
H=ftrans2(B,T)
```

其中，B 是一维 FIR 滤波器，T 是将一维 FIR 滤波器变换为二维 FIR 滤波器 H 的频率变换矩阵，缺省值为[1 2 1; 2 −4 2; 1 2 1]/8。例如，以下代码首先设计一个一维最优波纹 FIR 滤波器，然后使用这个滤波器创建一个与之具有相同特征的二维 FIR 滤波器：

```
b = remez(10,[0 0.4 0.6 1],[1 1 0 0]);          %一维最优波纹滤波器设计
h = ftrans2(b);
[H,w] = freqz(b,1,64,'whole');
colormap(jet(64))
subplot(1,2,1),plot(w/pi-1,fftshift(abs(H)))
subplot(1,2,2), freqz2(h,[32 32])
```

一维和二维最优波纹 FIR 滤波器的频率响应曲线分别如图 5.13(*a*)和 5.13(*b*)所示。



(*a*) (*b*)

图 5.13 最优波纹 FIR 滤波器的一维和二维频率响应曲线

(*a*) 一维响应曲线；(*b*) 二维响应曲线

以上代码中的 freqz2 函数是用来计算二维滤波器频率响应的，该函数的调用格式如下：

```
[H,f1,f2] = freqz2(h);
```

其中，H 为频率响应矩阵，f1 和 f2 为相应的频率点向量。freqz2 函数自动对 f1 和 f2 的频率进行规格化，使原始数值 1.0 或弧度 $\pi$ 对应于采样频率的一半。对于简单的 m×n 响应，freqz2 函数使用二维快速傅立叶变换函数 fft2 计算频率响应；如果需要计算任意频率点向量的响应，那么 freqz2 函数将根据傅立叶变换定义计算响应。如果不指定 freqz2 函数的输出参数，那么该函数将自动生成一个频率响应的网格图形。例如，以下滤波器的网格图形如图 5.14 所示：

```
h =[0.1667 0.6667 0.1667
    0.6667 -3.3333 0.6667
    0.1667 0.6667 0.1667];
freqz2(h)
```

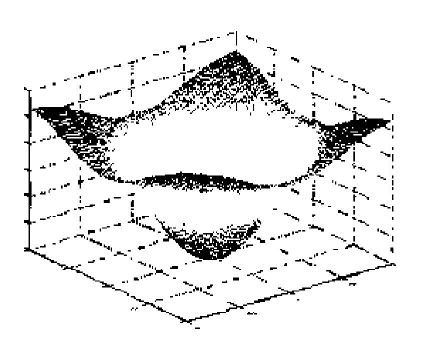


图 5.14 频率响应网格图形示意图

通过以上的例子可以看出，利用函数 ftrans2 就可以使用所有信号处理和滤波器设计工具箱中的设计函数进行各种二维图像滤波器的设计。例如，先调用函数 butter 进行一维低通、高通、带通或带阻巴特沃斯滤波器设计，然后将其变换为二维低通、高通、带通或带阻巴特沃斯滤波器(参见有关 MATLAB 信号处理和滤波器书籍)。

### 5.4.3 频率采样方法

频率采样方法是一种根据所需频率响应创建滤波器的方法。给出一个指定频率响应幅值的点阵，频率采样方法将创建一个相应的滤波器，该滤波器的频率响应将经过所有给定点。频率采样对于给定点之间的频率响应行为不做任何限制，因而给定点之间的频率响应通常都是振动的。

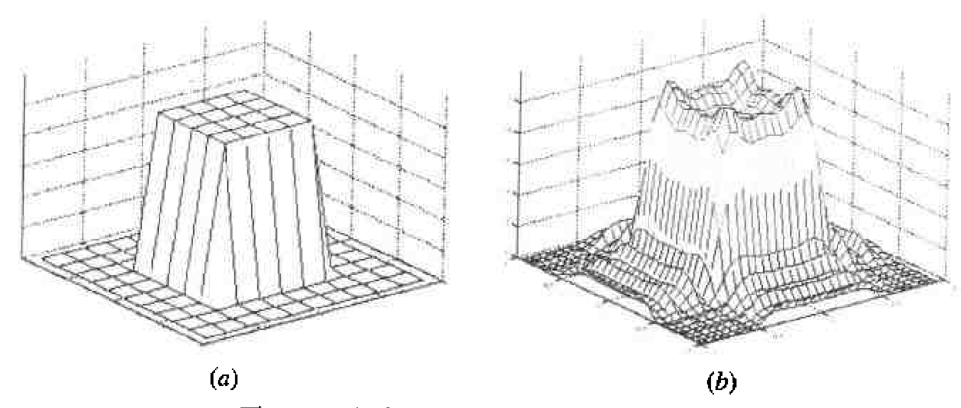
MATLAB 图像处理工具箱函数 fsamp2 可以实现二维 FIR 滤波器的频率采样设计。fsamp2 函数的调用格式如下：

```
H = fsamp2(F1,F2,HD,[M N])
```

其中，返回的 M×N 维滤波器 H 的频率响应将与频率响应 HD 中每一个由 F1 和 F2 指定的点相匹配。参数 F1、F2 和[M N]都是可选参数，缺省情况下 H 将与 HD 具有相同的大小，其频率响应将匹配 HD 中的每一个点。例如，以下代码将使用 fsamp2 函数创建一个 11×11 的滤波器，同时绘制所得滤波器的频率响应：

```
Hd = zeros(11,11); Hd(4:8,4:8) = 1;
[f1,f2] = freqspace(11,'meshgrid');
subplot(1,2,1),mesh(f1,f2,Hd), colormap(jet(64))
h = fsamp2(Hd);
subplot(1,2,2), freqz2(h,[32 32]), axis([-1 1 -1 1 0 1.2])
```

给定的频率响应如图 5.15(*a*)所示，设计所得滤波器的二维频率响应如图 5.15(*b*)所示。



(*a*) (*b*)

图 5.15 频率采样滤波器设计方法示意图

(*a*) 给定频率响应；(*b*) 所得滤波器的二维频率响应

也可以使用 freqspace 函数创建一个符合要求的频率响应幅值矩阵，该函数对任意大小的响应都能够返回均匀间隔的频率值。例如，以下代码将创建一个截止频率为 0.5 的圆形理想低通频率响应(参见图 5.16)：

```
[f1,f2] = freqspace(25,'meshgrid');
Hd = zeros(25,25); d = sqrt(f1.^2 + f2.^2) < 0.5;
Hd(d) = 1;
mesh(f1,f2,Hd)
```

将所需的频率响应离散化后调用函数 fsamp2 就可以实现任意所需的滤波器。注意，当所需频率响应中存在尖锐跃迁的时候，真实的频率响应就会出现振动现象。振动是频率采样设计中需要处理的基本问题，可以通过使用一个带宽较大的滤波器来减少振动的空间范围，但是，大滤波器不会降低振动的高度，而且进行滤波时将需要更多的计算时间。如果希望获得近似于所需频率响应的光滑结果，可以考虑采用频率变换方法和窗口方法。

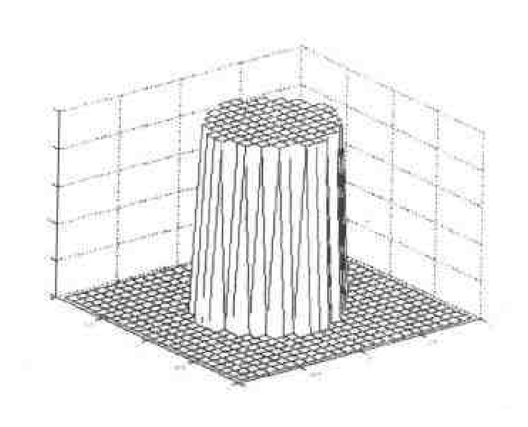


图 5.16 创建所需频率响应矩阵示意图

### 5.4.4　窗口方法

窗口方法是将理想的脉冲响应与窗口函数相乘来获得滤波器的方法。和频率采样方法类似，窗口方法将产生一个频率响应近似于所需频率响应的滤波器，但是窗口方法得到的结果比频率采样方法要好。

MATLAB 图像处理工具箱提供两个函数实现基于窗口的滤波器设计：fwind1 和 fwind2。fwind1 函数根据由输入参数指定的一维窗口创建一个二维窗口，然后进行二维滤波器设计，fwind2 函数则直接使用指定的二维窗口函数设计二维滤波器。

fwind1 函数支持两种不同的二维窗口创建方法：一种方法是对一个单独的一维窗口进行变换，使用类似旋转的方法创建一个近似中心对称的二维窗口，这种方法的调用格式如下：

```
H=fwind1(HD,WIN)
```

其中，HD 是所需的频率响应，WIN 是指定的一维窗口函数，可以是任意一种信号处理工具箱中的窗口函数，例如，oxcar、hamming、hanning、kaiser 等。返回的滤波器 H 是一个行数等于 WIN 长度的方阵。例如，以下代码将根据所需的频率响应 Hd（如图 5.15(*a*)所示）创建一个 11×11 的滤波器，其中 hamming 函数是一个能够创建一维窗口的信号处理工具箱函数，fwind1 函数将这个一维窗口拓展为一个二维窗口，所得滤波器的二维频率响应曲线如图 5.17(*a*)所示：

```
Hd = zeros(11,11); Hd(4:8,4:8) = 1;
[f1,f2] = freqspace(11,'meshgrid');
h = fwind1(Hd,hamming(11));
subplot(1,2,1), freqz2(h,[32 32])
```

fwind1 函数另一种创建二维窗口的方法是计算两个一维窗口的乘积，从而生成一个二维矩形可分窗口，其调用格式如下：

```
H = fwind1(HD,WIN1,WIN2)
```

例如，以下代码将根据以上的 Hd 创建二维窗口，所得的滤波器响应如图 5.17(*b*)所

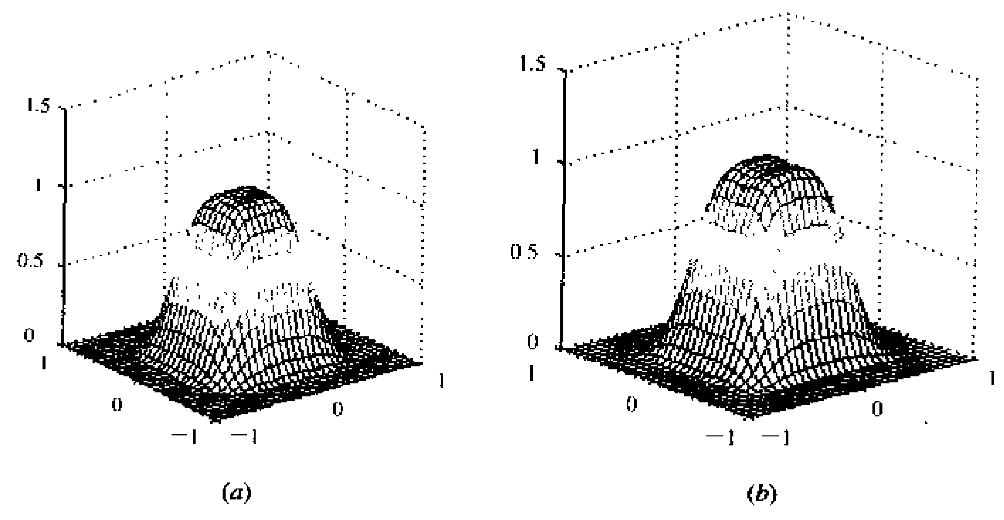



图 5.17　两种 fwind1 窗口方法创建的滤波器响应曲线

(*a*) 第一种创建方式所得的频率响应曲线；(*b*) 第二种创建方式所得的频率响应曲线

示。从图中可以看出，这两种窗口方法创建的滤波器频率响应曲线有着细微的不同。

```
h1 = fwind1(Hd,hamming(11), hanning(11));
subplot(1,2,2), freqz2(h2,[32 32])
```

fwind2 函数的调用格式如下：

```
H = fwind2 (F1,F2,HD,WIN)
```

其中，HD 为所需的频率响应，WIN 为指定的窗口。F1 和 F2 是可选参数，用来指定在 x 轴和 y 轴任意位置处的所需频率响应。F1 和 F2 都是取介于－1.0～1.0 之间的实数，数值 1.0 对应于半周期的长度。

以下代码将使用 fwind2 函数创建一个近似圆对称的带通滤波器，通频带为 0.1～0.5（规格化频率），输出滤波器的频率响应曲线如图 5.18(c)所示。

```
[f1,f2] = freqspace(21,'meshgrid');
Hd1 = ones(21);
r = sqrt(f1.^2 + f2.^2);
Hd1((r<0.1) | (r>0.5)) = 0;
win = fspecial('gaussian',21,2);
win = win ./ max(win(:));          % 使最大的窗口值为 1
h3 = fwind2(Hd1,win);
figure,freqz2(h3)
```

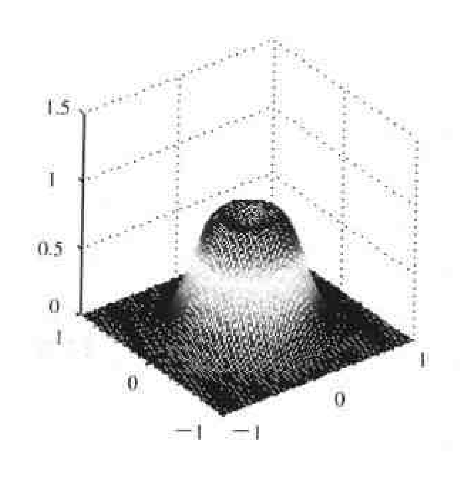



图 5.18　窗口方式设计所得滤波器的二维频率响应曲线

## 【习　题】

1. 证明相关性定理。
2. 利用 MATLAB 设计一个低通滤波器和一个高通滤波器，并绘制其频率响应曲线。

# 第六章 二值形态学操作

**本章要点：**

★ 二值形态学基本运算

★ 膨胀和腐蚀的 MATLAB 实现方法

★ 形态操作应用

★ 二进制图像的形态学应用

## 6.1　二值形态学基本运算

### 6.1.1　二值形态学概念

最初形态学是生物学中研究动物和植物结构的一个分支，后来也用数学形态学来表示以形态为基础的图像分析数学工具。形态学的基本思想是使用具有一定形态的结构元素来度量和提取图像中的对应形状，从而达到对图像进行分析和识别的目的。数学形态学可以用来简化图像数据，保持图像的基本形状特性，同时去掉图像中与研究目的无关的部分。使用形态学操作可以完成增强对比度、消除噪声、细化、骨架化、填充和分割等常用图像处理任务。

数学形态学的数学基础和使用的语言是集合论，其基本运算有四种：膨胀（或扩张）、腐蚀（或侵蚀）、开启和闭合，基于这些基本运算还可以推导和组合成各种数学形态学运算方法。二值形态学中的运算对象是集合，通常给出一个图像集合和一个结构元素集合，利用结构元素对图像进行操作。这里要注意，实际运算中所使用的两个集合不能看成是互相对等的：如果 $A$ 是图像集合，$B$ 是结构元素（$B$ 本身也是一个图像集合），形态学运算将使用 $B$ 对 $A$ 进行操作。结构元素是一个用来定义形态操作中所用到的邻域的形状和大小的矩阵，该矩阵仅由 0 和 1 组成，可以具有任意的大小和维数，数值 1 代表邻域内的像素，形态学运算都是对数值为 1 的区域进行的运算。

### 6.1.2　膨胀和腐蚀

膨胀的运算符为“$\oplus$”，图像集合 $A$ 用结构元素 $B$ 来膨胀，记作 $A\oplus B$，其定义为

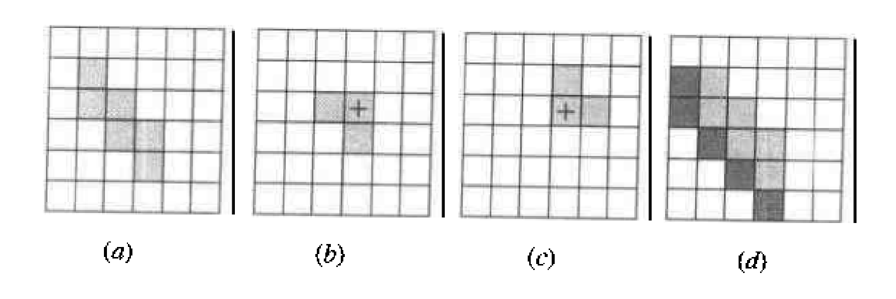
$$A \oplus B = \{x \mid [(\hat{B})_x \cap A] \neq \varnothing\} \tag{6.1}$$

其中，$\hat{B}$ 表示 $B$ 的映像，即与 $B$ 关于原点对称的集合。式(6.1)表明，用 $B$ 对 $A$ 进行膨胀的过程是这样的：首先对 $B$ 作关于原点的映射，再将其映像平移 $x$，当 $A$ 与 $B$ 映像的交集不为空集时，$B$ 的原点就是膨胀集合的像素。也就是说，用 $B$ 来膨胀 $A$ 得到的集合是 $\hat{B}$ 的

位移与 $A$ 至少有一个非零元素相交时 $B$ 的原点的位置集合。因而式(6.1)也可以写成：

$$A \oplus B = \{x \mid [(B)_x \cap \hat{A}] \subseteq A\} \tag{6.2}$$

如果将 $B$ 看成是一个卷积模板，膨胀就是对 $B$ 作关于原点的映像，然后再将映像连续地在 $A$ 上移动而实现的。图 6.1 给出了膨胀运算的一个示意图，其中“+”号表示原点【下同】。



(a)　(b)　(c)　(d)

图 6.1　膨胀运算示意图

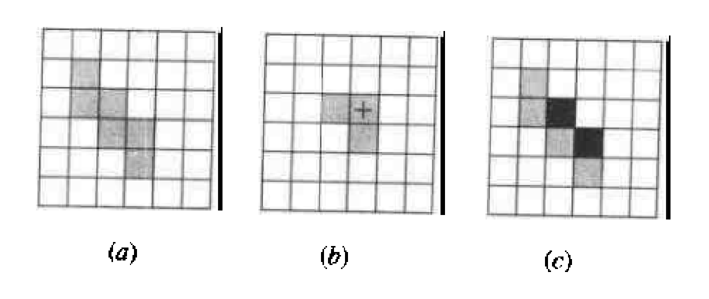
(a) 集合 A；(b) 集合 B；(c) B 的映像；(d) 膨胀结果

图 6.1(a)中的阴影部分为 $A$，图(b)中的阴影部分为 $B$，图(c)中的阴影部分为 $B$ 的映像，而图(d)中的阴影部分表示 $A \oplus B$，其中深色阴影表示图像膨胀后扩张的部分。

腐蚀的运算符是“$\ominus$”，$A$ 用 $B$ 来腐蚀记作 $A \ominus B$，其定义为

$$A \ominus B = \{x \mid (B)_x \subseteq A\} \tag{6.3}$$

式(6.3)表明，$A$ 用 $B$ 腐蚀的结果是所有满足将 $B$ 平移 $x$ 后，$B$ 仍全部包含在 $A$ 中的 $x$ 的集合，从直观上看就是 $B$ 经过平移后全部包含在 $A$ 中的原点组成的集合。图 6.2 所示为一个腐蚀运算的示意图。



(a)　(b)　(c)

图 6.2　腐蚀运算示意图

(a) 集合 A；(b) 集合 B；(c) 腐蚀结果

图 6.2(a)中的阴影部分为 $A$，图(b)中的阴影部分为 $B$，图(c)中的深色阴影部分表示 $A \ominus B$。

以上讨论中都假设原点是位于结构元素之中的，那么对于膨胀运算来说，总有

$$A \subseteq A \oplus B \tag{6.4}$$

而对于腐蚀运算来说，总有

$$A \ominus B \subseteq A \tag{6.5}$$

如果原点不在结构元素中，那么结果将有所变化。对于膨胀元素来说，总有

$$A \not\subset A \oplus B \tag{6.6}$$

图 6.3 所示为原点不包括在结构元素中时的膨胀运算示意图。图 6.3(d)的所有阴影表

示膨胀结果。从图中可以看出，标有叉号的点原来属于 $A$，但是现在并不属于膨胀结果。

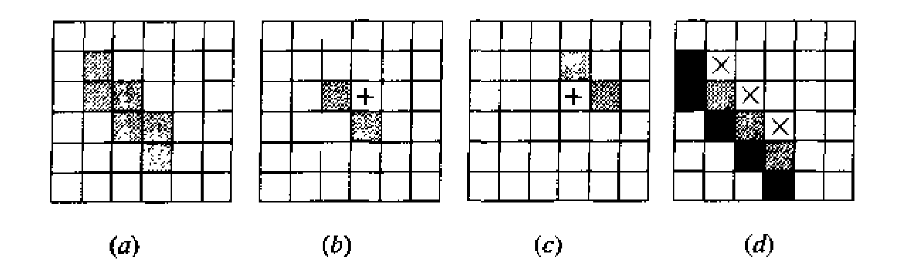


图 6.3　原点不包含在结构元素中时的膨胀运算示意图
(*a*) 集合 $A$；(*b*) 集合 $B$；(*c*) $B$ 的映像；(*d*) 膨胀结果

对于腐蚀运算来说，如果原点不包含在结构元素中，那么会有两种可能，一种是：

$$A \ominus B \subseteq A \tag{6.7}$$

另一种是：

$$A \ominus B \not\subset A \tag{6.8}$$

图 6.4 和图 6.5 分别给出了当原点不包括在结构元素中时以上两种情况的腐蚀运算示意图，图中的深色阴影部分表示腐蚀结果。

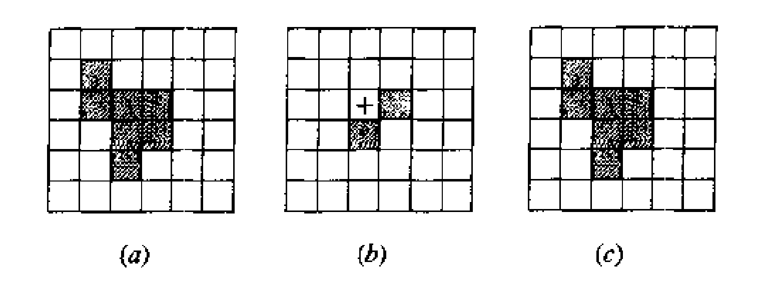


图 6.4　当原点不包含在结构元素中时腐蚀运算示意图一
(*a*) 集合 $A$；(*b*) 集合 $B$；(*c*) 腐蚀结果

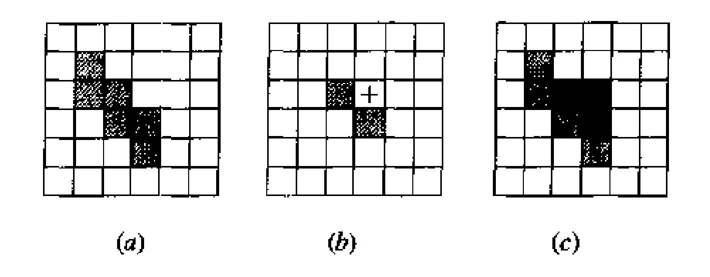


图 6.5　当原点不包含在结构元素中时腐蚀运算示意图二
(*a*) 集合 $A$；(*b*) 集合 $B$；(*c*) 腐蚀结果

### 6.1.3　膨胀和腐蚀的对偶性

膨胀和腐蚀这两种操作有着密切的关系：使用结构元素对图像进行腐蚀操作相当于使用该结构元素的映像对图像背景进行膨胀操作，反之亦然。这也就是说

$$(A \oplus B)^c = A^c \ominus B^c \tag{6.9}$$

$$A^c \oplus B^c = (A \ominus B)^c \tag{6.10}$$

膨胀和腐蚀的对偶性可从图 6.6 中体现出来。图 6.6(*a*)、(*b*)就是图 6.1 中的 *A* 及其膨胀结果，图 6.6(*c*)则是图 6.6(*a*)的背景图像，图 6.6(*d*)是使用图 6.1(*c*)所示的结构元素对图 6.6(*c*)进行腐蚀的结果，显然图 6.6(*d*)就是图(*b*)的背景图像。

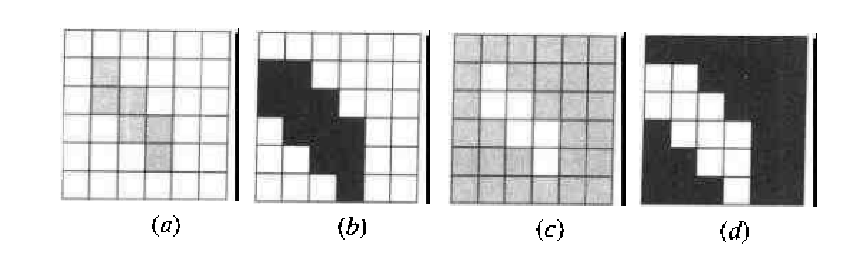


图 6.6　膨胀与腐蚀对偶性示意图

(*a*) 集合 *A*；(*b*) 腐蚀结果；(*c*) *A* 的背景图像；(*d*) 背景图像的腐蚀结果

### 6.1.4　开启与闭合

使用同一个结构元素对图像先进行腐蚀然后再进行膨胀的运算称为开启，先进行膨胀然后再进行腐蚀的运算则称为闭合，这两种运算也是形态学中的重要运算。

开启的运算符为"∘"，*A* 用 *B* 来开启记为 $A \circ B$，其定义如下：

$$A \circ B = (A \ominus B) \oplus B \tag{6.11}$$

闭合的运算符为"·"，*A* 用 *B* 来闭合记为 $A \cdot B$，其定义如下：

$$A \cdot B = (A \oplus B) \ominus B \tag{6.12}$$

开启和闭合不受原点位置的影响，无论原点是否包含在结构元素中，开启和闭合的结果都是一定的。根据膨胀和腐蚀的对偶性可知，开启与闭合运算也具有对偶性。

## 6.2　膨胀和腐蚀的 MATLAB 实现方法

### 6.2.1　图像处理的膨胀与腐蚀概念

在 MATLAB 图像处理工具箱中，膨胀一般是给图像中的对象边界添加像素，而腐蚀则是删除对象边界像素。在形态学的膨胀和腐蚀操作中，输出图像中所有给定像素的状态都是通过对输入图像中相应像素及其邻域使用一定的规则来确定的。进行膨胀操作时，输出像素值是输入图像相应像素邻域内所有像素的最大值。在二进制图像中，如果任何一个像素值为 1，那么对应的输出像素值为 1。而在腐蚀操作中，输出像素值是输入图像相应像素邻域内所有像素的最小值。在二进制图像中，如果任何一个像素值为 0，那么对应的输出像素值为 0。

图 6.7 说明了一幅二进制图像的膨胀规则。在图 6.7 中，形态膨胀运算对应的输出像素取值为 1，这是因为由结构元素定义的邻域中有一个元素数值为 1(即状态为 on)。图 6.8 说明了灰度图像的膨胀过程。

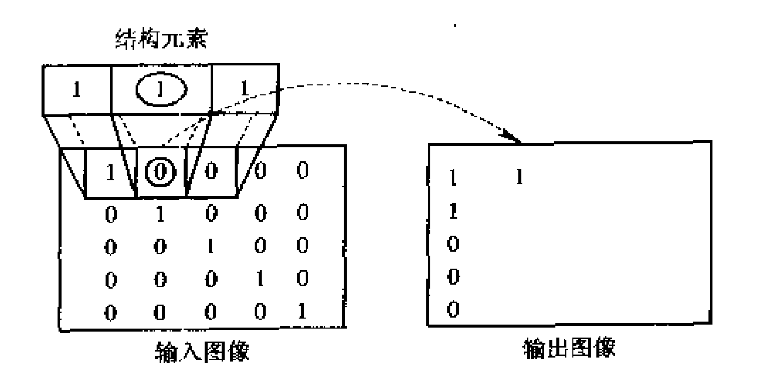



图 6.7　二进制图像的膨胀规则示意图

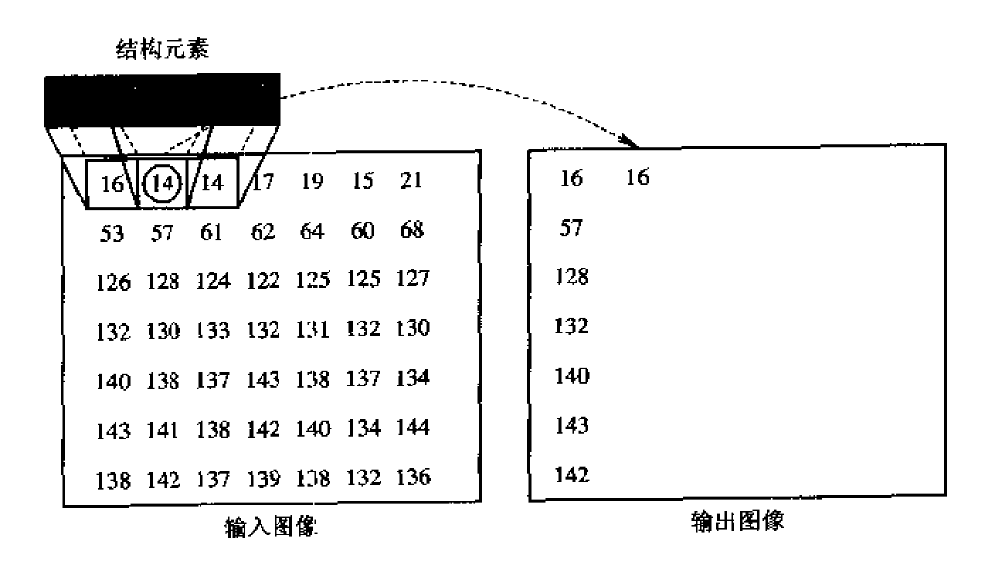



图 6.8　灰度图像的膨胀规则示意图

结构元素的原点都定义在对输入图像感兴趣的位置处。对于图像边缘的像素，由结构元素定义的邻域将会有一部分位于图像边界之外。为了处理边界像素，一般进行形态学运算的函数都会给超出图像、未指定数值的像素指定一个数值，这看起来好像是函数给图像填充了额外的行和列。膨胀和腐蚀操作的像素的填充值是不同的，对于二进制图像和灰度图像，膨胀和腐蚀操作使用的填充方法具体如下：

■ 膨胀：超出图像边界的像素值定义为该数据类型允许的最小值。对于二进制图像，这些像素值被设置为 0；对于灰度图像，uint8 类型的最小值也是 0；

■ 腐蚀：超出图像边界的像素值定义为该数据类型允许的最大值。对于二进制图像，这些像素值被设置为 1；对于灰度图像，uint8 类型的最大值是 255。

通过对膨胀操作使用最小值填充和对腐蚀操作使用最大值填充可以消除边界效应(输出图像靠近边界处的区域与图像其他部分不连续)，否则，如果腐蚀操作使用最小值进行填充，那么图像腐蚀将会导致输出图像围绕着一个黑色边框。

### 6.2.2　结构元素

膨胀和腐蚀操作的基本组成部分就是用来测试输入图像的结构元素。二维(平面)结构

元素由一个数值为 0 或 1 的矩阵组成，通常比待处理的图像小得多。结构元素的原点指定了图像中需要处理的像素范围，结构元素中数值为 1 的点决定了结构元素邻域中的像素在进行膨胀或腐蚀操作时是否需要参与计算。三维或非平面的结构元素使用 0 和 1 来定义结构元素在 x 和 y 平面上的范围，采用第三维来定义高度。

MATLAB 的形态函数使用以下函数来获得任意大小和维数的结构元素的原点坐标：

```
origin = floor((size(nhood)+1)/2)
```

在以上的语句中，nhood 是指结构元素定义的邻域。结构元素在 MATLAB 中被定义为一个称为 STREL 的对象，由于 MATLAB 规定不能在表达式中直接使用对象本身的大小，所以必须使用 STREL 对象的 nhood 属性来获得结构元素的邻域。图 6.9 给出了一个钻石形结构元素的例子。

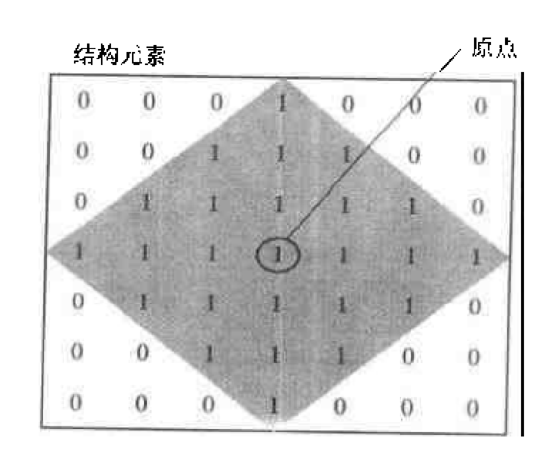



图 6.9　钻石形结构元素示意图

可以使用 MATLAB 图像处理工具箱函数 strel 来创建任意大小和形状的 STREL 对象。strel 函数支持许多种常用的形状，如线形(line)、钻石形(diamond)、圆盘形(disk)和球形(ball)等。例如，以下语句将创建一个平面钻石形结构元素：

```
se = strel('diamond',3)
```

观察 strel 返回结构元素 se 的数值可以看出 MATLAB 结构元素的有关信息：

```
se =0 0 0 1 0 0 0
    0 0 1 1 1 0 0
    0 1 1 1 1 1 0
    1 1 1 1 1 1 1
    0 1 1 1 1 1 0
    0 0 1 1 1 0 0
    0 0 0 1 0 0 0
```

**小技巧：**

通常可以选择一个与希望处理的输入图像相同形状的结构元素。例如，如果希望找到图像中的直线，那么就可以创建一个线形结构元素。

为了提高执行效率，strel 函数可能会将结构元素拆为较小的块，这种技术通常称为结构元素分解。例如，使用一个 11×11 正方形结构元素的膨胀操作可以首先用 1×11 的结构元素进行膨胀，然后再使用 11×1 的结构元素进行膨胀，这样做在理论上会使执行速度提高 6.5 倍。圆盘形和球形结构元素的分解结果是近似的，其他形状的分解结果是精确的。如果希望观察分解得到的结构元素序列，可调用函数 getsequence。getsequence 函数返回一个分解后的结构元素数组。例如，以下代码将获得一个钻石形结构元素分解所得的结构元素序列：

```
seq = getsequence(se);
seq(1)
ans =0 1 0
     1 1 1
     0 1 0
seq(2)
ans =0 1 0
     1 0 1
     0 1 0
seq(3)
ans =0 0 1 0 0
     0 0 0 0 0
     1 0 0 0 1
     0 0 0 0 0
     0 0 1 0 0
```

### 6.2.3 图像膨胀

MATLAB 使用 imdilate 函数进行图像膨胀。imdilate 函数需要两个基本输入参数：待处理的输入图像以及结构元素对象。结构元素可以是由 strel 函数返回的对象，也可以是一个定义结构元素邻域的二进制矩阵。imdilate 函数还可以接受两个可选的参数：PADOPT 和 PACKOPT。PADOPT 参数可以影响输出图像的大小，而 PACKOPT 参数用来说明输入图像是否为打包二进制图像。下面通过一个例子来说明如何使用 imdilate 函数进行图像膨胀。

**例 6.1**：对一个包含矩形对象的简单二进制图像进行膨胀操作。

首先创建一个包含矩形对象的二进制图像矩阵：

```
BW = zeros(9,10);
BW(4:6,4:7) = 1;
```

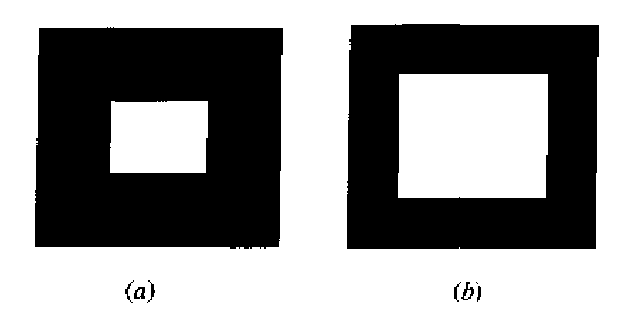
使用一个 3×3 的正方形结构元素对象对创建的图像进行膨胀：

```
SE = strel('square',3)
```

为了膨胀这幅图像，将图像 BW 和结构元素 SE 传递给 imdilate 函数：

```
BW2 = imdilate(BW,SE);
```

膨胀前、后的图像如图 6.10 所示。



(*a*)　　(*b*)

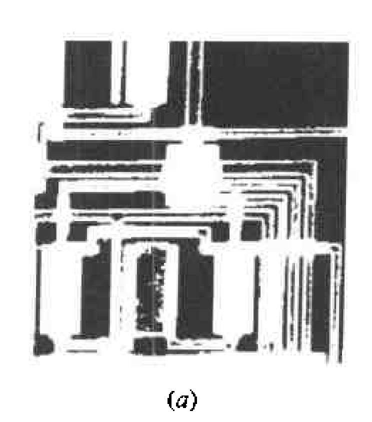
图 6.10　图像膨胀前、后的显示效果比较
(*a*) 膨胀前；(*b*) 膨胀后

### 6.2.4 图像腐蚀

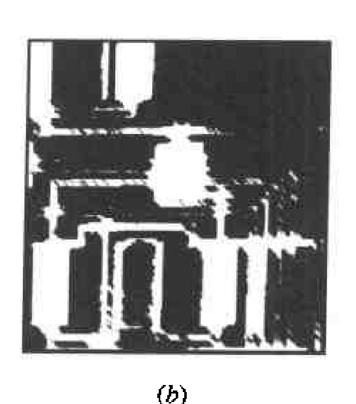
MATLAB 使用 imerode 函数进行图像腐蚀。imerode 函数需要两个基本输入参数：待处理的输入图像以及结构元素对象。imerode 函数还可以接受三个可选参数：PADOPT、PACKOPT 和 M，前两个参数的含义与 imdilate 函数的可选参数类似，另外，如果图像是

打包二进制图像，那么 M 将指定原始图像的行数。下面通过一个例子说明如何使用 imerode 函数进行图像的腐蚀操作。

**例 6.2**：对图像 circbw.tif(如图 6.11(*a*)所示)进行腐蚀操作。



(*a*)



(*b*)

图 6.11　图像 circbw.tif 的腐蚀前、后显示效果比较
(*a*) 原图像(腐蚀前)；(*b*) 腐蚀后

首先，将图像读入 MATLAB 工作平台中：

```
BW1 = imread('circbw.tif');
```

其次，创建一个任意形状的结构元素对象：

```
SE = strel('arbitrary',eye(5));
SE= 1 0 0 0 0
    0 1 0 0 0
    0 0 1 0 0
    0 0 0 1 0
    0 0 0 0 1
```

最后，以图像 BW1 和结构元素 SE 为参数调用 imerode 函数进行腐蚀操作：

```
BW2 = imerode(BW1,SE);
imshow(BW1)
figure, imshow(BW2)
```

腐蚀后的图像如图 6.11(*b*)所示。输出图像右侧的钻石条纹是由结构元素的形状导致的。

## 6.2.5　综合使用膨胀和腐蚀操作

在图像处理操作中经常综合使用膨胀和腐蚀两种操作(例如，图像的开启和闭合)。下面将以图像开启为例说明如何综合使用 imdilate 和 imerode 函数以实现图像处理操作(事实上 MATLAB 图像处理工具箱已经提供了 imopen 函数来实现对图像的开启操作)。

**例 6.3**：从图像 circbw.tif(如图 6.12(*a*)所示)中删除所有电流线，仅保留芯片对象。

开启操作在删除图像中的小对象的同时具有保持图像中大对象的尺寸和形状不变的能力。图像的开启操作可以分以下步骤进行：

(1) 创建结构元素。结构元素必须具有适当的大小，既可以删除电流线，又不足以删除矩形：

```
SE = strel('rectangle',[40 30]);
```

(2) 使用结构元素腐蚀图像。

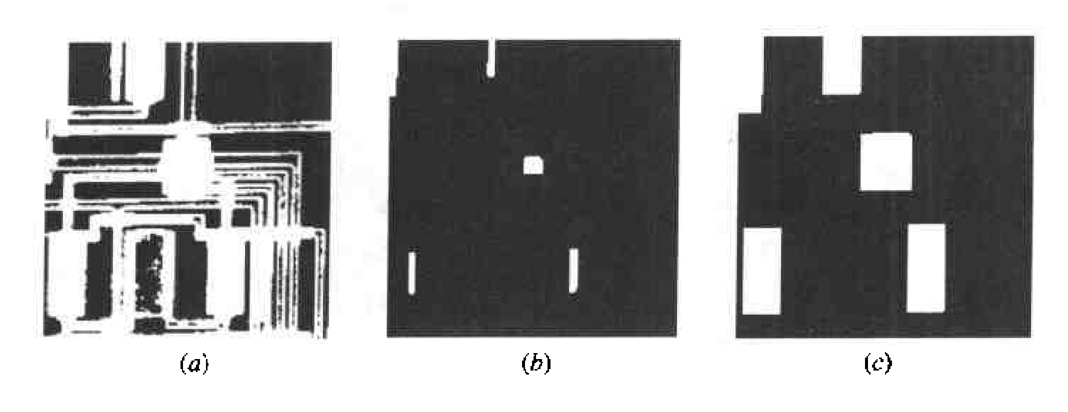
```
BW1 = imread('circbw.tif');
BW2 = imerode(BW1,SE);
imshow(BW2)
```

这一步的操作将删除所有直线，但是也会缩减矩形，如图 6.12(*b*)所示。

(3) 恢复矩形为原有大小。使用相同的结构元素对腐蚀过的图像进行膨胀：

```
BW3 = imdilate(BW2,SE);
imshow(BW3)
```

最终的开启结果如图 6.12(*c*)所示。



(*a*)　(*b*)　(*c*)

图 6.12　图像的膨胀、腐蚀综合操作的显示效果
(*a*) 原图像；(*b*) 腐蚀后；(*c*) 膨胀后

### 6.2.6　基于膨胀和腐蚀的形态操作

MATLAB 图像处理工具箱中基于膨胀和腐蚀的形态操作函数主要有：

■ bwhitmiss：图像的逻辑"与"操作。该函数使用一个结构元素对图像进行腐蚀操作后，再使用第二个结构元素对图像进行腐蚀操作。其调用格式如下：

```
BW2 = bwhitmiss (BW,SE1,SE2)
```

其中，BW 为二进制图像，SE1 和 SE2 分别为结构元素。

■ imbothat：从原始图像中减去经过形态关闭后的图像。该函数可以用来寻找图像中的灰度槽，其调用格式如下：

```
IM2 = imbothat (IM,SE)
```

其中，IM 为输入图像，SE 为结构元素。

■ imclose：闭合操作。该函数首先对图像进行膨胀，然后再对膨胀后的图像进行腐蚀，两个操作使用同样的结构元素。其调用格式如下：

```
IM = imclose (IM,SE)
```

■ imopen：开启操作。该函数首先对图像进行腐蚀，然后再进行膨胀。两个操作使用相同的结构元素。其调用格式如下：

```
IM = imopen (IM,SE)
```

■ imtophat：从原始图像中减去形态开启后的图像，可以用来增强图像的对比度。其调用格式如下：

```
IM = imtophat (IM,SE)
```

MATLAB 工具箱还有一些较为复杂的形态复合操作，例如 bwmorph、bwperim 等。

下面通过两个例子来说明这些操作函数的使用方法及其在图像处理中的作用。

**例 6.4**：将图 6.12(*a*)进行骨架化。

为了将图像中的所有对象简化为线条，但不修改图像的基本结构，可使用 bwmorph 函数(参见 MATLAB 在线帮助)，这个过程被称为骨架化。

```
BW1 = imread('circbw.tif');
BW2 = bwmorph(BW1,'skel',Inf);
imshow(BW1)
figure, imshow(BW2)
```

骨架化的结果如图 6.13 所示。

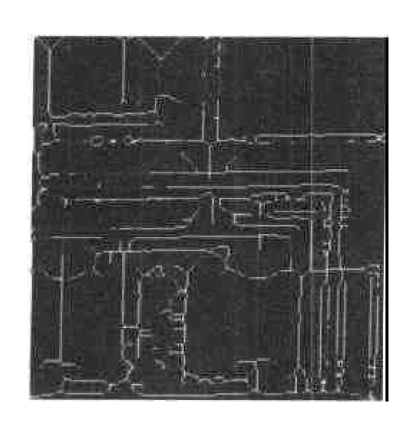


图 6.13　图像 circbw.tif 的骨架化效果



图 6.14　图像 circbw.tif 的边界像素效果

**例 6.5**：寻找图 6.12(*a*)中的边界像素。

如果一个像素满足：像素状态为 on 而且该像素邻域中有一个或多个像素状态为 off，那么就认为该像素为边界像素。bwperim 函数可以判断二进制图像中对象的边界像素：

```
BW1 = imread('circbw.tif');
BW2 = bwperim(BW1);
imshow(BW1)
figure, imshow(BW2)
```

找到的边界像素如图 6.14 所示。

# 6.3　形态操作应用

## 6.3.1　形态重构

形状重构是图像形态处理的重要操作之一，通常用来强调图像中与掩模图像指定对象相一致的部分，同时忽略图像中的其他对象。形态重构根据一幅图像(称之为掩模图像)的特征对另一幅图像(称之为标记图像)进行重复膨胀，重点是要选择一个合适的标记图像，使膨胀所得的结果能够强调掩模图像中的主要对象。每一次膨胀处理从标记图像的峰值点开始，整个膨胀过程将一直重复，直到图像的像素值不再变化为止。

形态重构操作具有这样一些独有的特性：形态重构处理是基于两幅图像的，一个是标记图像，另一个是掩模图像，而不仅仅是一幅图像和一个结构元素；重构将一直重复直至图像稳定(即图像不再变化)；形态重构是基于连通性概念的，而不是基于结构元素的。

图 6.15 说明了一维图像形态重构的过程。从图 6.15(*a*)中可以看出，每一次成功的膨胀都是限制在掩模下的。当膨胀操作不会再改变图像数值时，重构过程停止，最后一次的膨胀结果就是重构图像(如图 6.15(*b*)所示)。

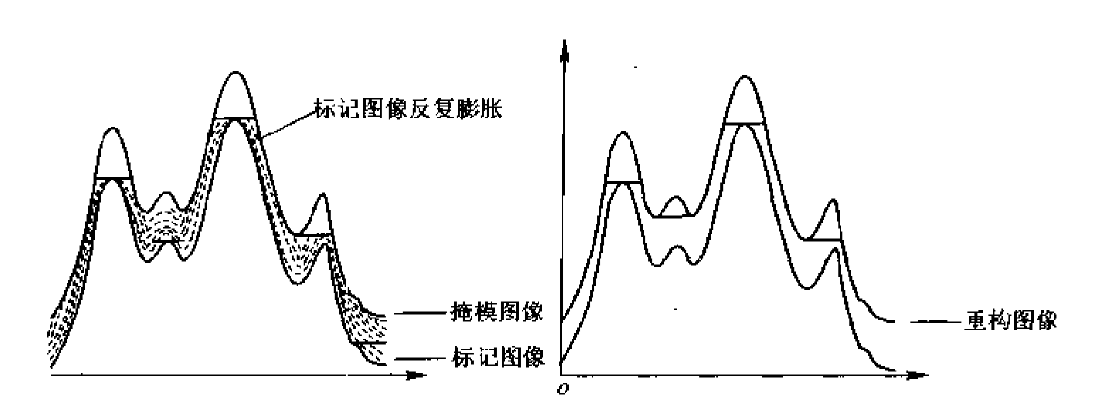



图 6.15 一维图像形态重构过程及结果示意图
(*a*) 一维图像形态重构过程；(*b*) 最终的形态重构结果

形态重构从标记图像的峰值开始，根据像素的连通性依次计算到图像的其他部分。像素的连通性定义了该像素是与哪些像素相连的，即像素的邻域是由哪些像素构成的。用户选择的邻域类型将会影响图像中所能找到的对象数目和对象边界。许多形态操作的结果通常因指定的连通类型不同而不同。例如，图 6.16 所示的二进制图像包含一个像素值为 1 的对象。

```
0  0  0  0  0  0  0  0
0  1  1  1  1  1  0  0
0  1  0  0  0  1  0  0
0  1  0  0  0  1  0  0
0  1  0  0  0  1  0  0
0  1  1  1  1  1  0  0
0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0
```

图 6.16 像素连通性示例

在图 6.16 中，如果对象是 4 连通的，那么就不会认为这些数值为 1 的像素构成了一个对象，图像全部都是由背景像素构成的，因而处理时会将所有元素数值设置为 0；如果对象是 8 连通的，那么数值为 1 的像素将构成一个环形对象，这就使得图像不但包含环形对象，还包括两个分离的背景对象：闭环中的对象和闭环外的对象。表 6.1 列出了 MATLAB 图像处理工具箱支持的所有标准二维和三维连通类型。

还可以使用一个数值为 0 和 1 的 3×3×…×3 数组来指定自定义的邻域，数值 1 可以指定相对于中心元素的邻域连通性。可以对三维图像使用二维连通性，连通性能够作用于三维图像的每一"页"。例如，在 MATLAB 中使用以下数组将定义一个能够将图像分割为列的南/北连通性(即只有位于像素正上方或正下方的像素才被认为在该像素的邻域内)：

CONN = [ 0 1 0; 0 1 0; 0 1 0 ]

⚠ **注意:**

自定义的连通性数组必须按中心元素对称。

⚠

**表 6.1 MATLAB 标准二维和三维连通类型**

| 维数 | 连通性 | 说明 | 图例 |
|---|---|---|---|
| 二维 | 4 连通 | 边缘相连的像素为连通像素，即只有水平或垂直方向相连的相邻像素对才被视为同一对象 | |
| | 8 连通 | 边缘或角相连的像素为连通像素，即水平、垂直或对角方向相连的相邻像素都被视为同一对象 | |
| 三维 | 6 连通 | 面相连的像素为连通像素 | |
| | 18 连通 | 面或边相连的像素为连通像素 | |
| | 26 连通 | 面、边或角相连的像素为连通像素 | |

下面说明 MATLAB 形态重构的实现方法。考虑图 6.17 所示的简单图像矩阵，该图像包含两个主要对象，这两个对象的像素块分别由数值 14 和 18 组成，图像的大部分背景像素值为 10，其他为 11。

```
A = [ 10  10  10  10  10  10  10  10  10  10;
      10  14  14  14  10  10  11  10  11  10;
      10  14  14  14  10  10  10  11  10  10;
      10  14  14  14  10  10  11  10  11  10;
      10  10  10  10  10  10  10  10  10  10;
      10  11  10  10  10  18  18  18  10  10;
      10  10  10  11  10  18  18  18  10  10;
      10  10  11  10  10  18  18  18  10  10;
      10  11  10  11  10  10  10  10  10  10;
      10  10  10  10  10  10  11  10  10  10];
```

图 6.17 形态重构的图像矩阵示例

首先要创建标记图像。正如在膨胀和腐蚀中使用构造结构元素一样，标记图像的特征能够决定形态重构结果所具有的特征，所以标记图像的峰值应该确定掩模图像中希望强调对象的位置。MATLAB 创建标记图像的一种方法是使用 imsubtract 函数将掩模图像减去

一个常数，例如：

```
marker = imsubtract(A,2)
```

然后调用 imreconstruct 函数进行图像的形态重构。imreconstruct 函数的调用格式如下：

```
IM =imreconstruct (MARKER,MASK,CONN)
```

其中，MARKER 和 MASK 分别表示标记图像和掩模图像，第三个输入参数 CONN 是可选的，用来指定连通类型，缺省情况下对二维图像使用 8 连通，对三维图像使用 26 连通。此处调用以下语句进行形态重构：

```
recon = imreconstruct(marker, mask)
```

重构结果如图 6.18 所示，从重构结果可以看出，形态重构所得的图像仅仅包含原始图像的两个主要对象，其他次要对象都被省略了。

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| 10 | 12 | 12 | 12 | 10 | 10 | 10 | 10 | 10 | 10 |
| 10 | 12 | 12 | 12 | 10 | 10 | 10 | 10 | 10 | 10 |
| 10 | 12 | 12 | 12 | 10 | 10 | 10 | 10 | 10 | 10 |
| 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| 10 | 10 | 10 | 10 | 10 | 16 | 16 | 16 | 10 | 10 |
| 10 | 10 | 10 | 10 | 10 | 16 | 16 | 16 | 10 | 10 |
| 10 | 10 | 10 | 10 | 10 | 16 | 16 | 16 | 10 | 10 |
| 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |

图 6.18　标记图像的形态重构结果

### 6.3.2　填充操作

填充操作是一种根据像素边界求取像素区域的操作，也是形态学的一种常用操作。下面通过介绍 MATLAB 的填充操作来说明怎样使用形态操作实现图像的填充操作。

MATLAB 的图像处理工具箱函数 imfill 可以用来实现灰度图像和二进制图像的填充操作。对于二进制图像，imfill 函数将相邻的背景像素(数值为 0)设置为对象的边界像素(数值为 1)；对于灰度图像，imfill 函数将被较亮区域围绕的黑暗区域的灰度值设置为与围绕区域的像素值相同的数值。从效果上来看，imfill 函数将删除没有连接到边界的局部极小值。这个操作在删除图像中的人为痕迹是非常有用的。imfill 函数有许多种调用格式，最常用的格式如下：

```
BW2 = imfill(BW1,LOCATIONS)
```

其中，BW1 为输入二进制图像，LOCATIONS 表示填充的起始点。

实现填充操作主要通过三个步骤：首先要指定填充操作的连通性；然后指定二进制图像填充的起始点；最后进行二进制图像和灰度图像区域的填充。像素的连通类型与形态重构中介绍的类型是一致的，可以通过 imfill 函数的输入参数指定，缺省情况下二维图像使用 4 连通，三维图像使用 6 连通。

对于二进制图像，可以通过传递位置下标或使用 imfill 函数交互模式用鼠标来指定填充操作的起始点。例如，如果指定像素 BW(4,3)作为起始点，那么调用 imfill 函数将仅仅

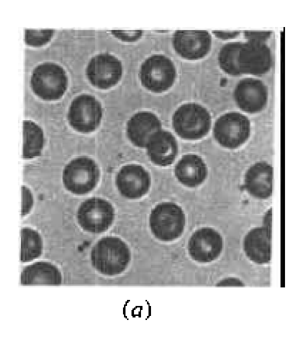
填充闭环内部的区域，因为缺省情况下背景是 4 连通的。

```
imfill(BW,[4 3])
ans = 0 0 0 0 0 0 0 0
      0 1 1 1 1 1 0 0
      0 1 1 1 1 1 0 0
      0 1 1 1 1 1 0 0
      0 1 1 1 1 1 0 0
      0 1 1 1 1 0 0 0
      0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0
```

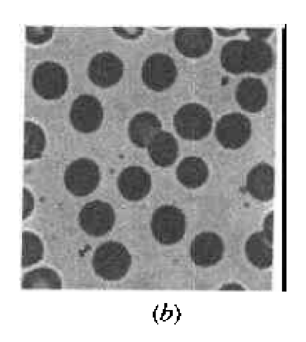
如果指定相同的起始点，但是使用 8 连通类型，那么 imfill 函数将填充整幅图像：

```
imfill(BW,[4 3],8)
ans = 1 1 1 1 1 1 1 1
      1 1 1 1 1 1 1 1
      1 1 1 1 1 1 1 1
      1 1 1 1 1 1 1 1
      1 1 1 1 1 1 1 1
      1 1 1 1 1 1 1 1
      1 1 1 1 1 1 1 1
      1 1 1 1 1 1 1 1
```

填充操作常常用来填充一幅图像的“洞”。例如，假设有一幅包含多个球体对象的二进制或灰度图像，在二维图像中球体对象将显示为圆盘形，但是由于原始照片的反射现象导致部分球体显示为环形。在对图像进行进一步的处理之前，首先要将环形洞填充起来。图 6.19 给出了图像进行洞填充前、后的效果对比。



(*a*)



(*b*)

图 6.19　洞填充前、后图像显示效果比较

(*a*) 填充前；(*b*) 填充后

### 6.3.3　图像的极值处理方法

如果将灰度图像视为三维图像，其中 x 和 y 轴描述像素位置，而 z 轴表示每一个像素的亮度，那么在这种理解下，灰度值就可以代表地图中的高度值，图像的高灰度值和低灰度值处就相当于地图的峰和谷。通常图像的峰、谷都代表相关的图像对象，具有重要的形态特征。例如，在一幅包含多个球形对象的图像中，高灰度的地方可以表示对象的顶部。

使用形态操作对图像的极值进行处理可以辨识图像中的对象。

图像中可以有多个局部极小值和极大值，但是只能有一个全局极大值和极小值。判断图像的峰和谷可以用来重建形态重构的标记图像。图 6.20 说明了一维图像极大值和极小值的概念。在一维图像中，x 代表像素位置，y 代表亮度，x 和 y 应理解为离散值，分别为像素单位和亮度单位，一般情况下可不标单位。

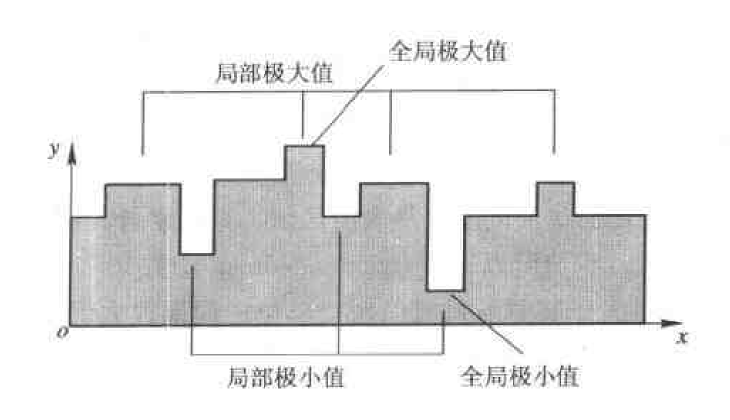



图 6.20　一维图像的极值概念

MATLAB 图像处理工具箱提供函数 imregionalmax 和 imregionalmin 来确定图像的所有局部极大值和极小值，另外，还提供 imextendedmax 和 imextendedmin 函数来确定所有大于或小于指定值的局部极大值和极小值。这些函数以灰度图像作为输入参数，返回一个二进制图像。在输出的二进制图像中，局部极大值或极小值被设置为 1，其他像素则被设置为 0。这些函数的调用格式分别如下：

```
BW = imregionalmax (I,CONN)
BW = imregionalmin (I,CONN)
BW = imextendedmax (I,H,CONN)
BW = imextendedmin (I,H,CONN)
```

其中，I 表示输入图像，CONN 表示连通类型，H 为一个非负标量，表示搜索阈值。

例如，图 6.21 所示的简单图像中包含数值为 13 和 18 局部极大值的两个像素块，其他一些局部极大值为 11。

```
A = [ 10  10  10  10  10  10  10  10  10  10;
      10  13  13  13  10  10  11  10  11  10;
      10  13  13  13  10  10  10  11  10  10;
      10  13  13  13  10  10  11  10  11  10;
      10  10  10  10  10  10  10  10  10  10;
      10  11  10  10  10  18  18  18  10  10;
      10  10  10  11  10  18  18  18  10  10;
      10  10  11  10  10  18  18  18  10  10;
      10  11  10  11  10  10  10  10  10  10;
      10  10  10  10  10  10  11  10  10  10];
```

图 6.21　包含多个局部极大值的图像示意图

imregionalmax 函数返回的二进制图像将会查明所有局部极大值(参见图 6.22)：

```
B = imregionalmax(A)
```

```
B =
     0     0     0     0     0     0     0     0     0     0
     0     1     1     1     0     0     1     0     1     0
     0     1     1     1     0     0     0     1     0     0
     0     1     1     1     0     0     1     0     1     0
     0     0     0     0     0     0     0     0     0     0
     0     1     0     0     0     1     1     1     0     0
     0     0     0     1     0     1     1     1     0     0
     0     0     1     0     0     1     1     1     0     0
     0     1     0     1     0     0     0     0     0     0
     0     0     0     0     0     0     1     0     0     0
```

图 6.22　返回所有局部极大值的二进制图像

有时可能仅仅需要找到那些灰度变化最大的区域，即像素与其邻域像素间的灰度变化大于一个固定阈值的区域，我们可以通过 imextendedmax 函数来实现这一功能。例如，如果希望找到图像 A 中数值大于其邻域像素值两个单位的局部极大值，使用 imextendedmax 函数可得到查询结果(如图 6.23 所示)：

```
B = imextendedmax(A,2)
```

```
B =
     0     0     0     0     0     0     0     0     0     0
     0     1     1     1     0     0     0     0     0     0
     0     1     1     1     0     0     0     0     0     0
     0     1     1     1     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     1     1     1     0     0
     0     0     0     0     0     1     1     1     0     0
     0     0     0     0     0     1     1     1     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
```

图 6.23　返回所有大于阈值为 2 的局部极大值的图像

在一幅图像中，每一个小灰度波动都代表一个局部极大值或极小值。用户可能仅对那些有重要意义的极大值或极小值感兴趣，而忽略由于背景纹理导致的较小的极小值或极大值。为了消除这些小的极小值和极大值，同时又要保证重要的极大值和极小值不变，可使用 imhmax 或 imhmin 函数。使用这些函数可以指定一个固定的标准或阈值 h，从而压制所有其他高度小于 h 的极大值或大于 h 的极小值。imhmax 和 imhmin 函数的调用格式分别如下，其中的参数含义与以上介绍的函数相同。

```
I2 = imhmax (I,H,CONN)
I2 = imhmin (I,H,CONN)
```

例如，图 6.17 所示的简单图像包含两个主要的局部极大值(包含数值 14 和 18 的像素块)和一些其他极大值块(数值为 11)。为了删除除两个主要极大值以外的其他所有的局部极大值，可使用函数 imhmax，指定阈值为 2。注意，imhmax 仅仅对极大值产生影响，对其他像素值不起作用。从图 6.24 所得的结果中可以看出，两个重要的极大值尽管数值减小了，但还是被保留下来了。

```
B = imhmax(A,2)
```

图 6.25 抽取了图 6.21 第二行的操作过程来说明 imhmax 函数是如何改变图像极

值的。

**⚠ 注意：**

imregionalmin、imregionalmax、imextendedmin 和 imextendedmax 函数返回另外一幅标记了图像局部极小值和极大值的二进制图像，而 imhmax 和 imhmin 函数则返回一幅直接在原始图像上进行修改后的图像。

B =

| 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 12 | 12 | 12 | 10 | 10 | 10 | 10 | 10 | 10 |
| 10 | 12 | 12 | 12 | 10 | 10 | 10 | 10 | 10 | 10 |
| 10 | 12 | 12 | 12 | 10 | 10 | 10 | 10 | 10 | 10 |
| 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| 10 | 10 | 10 | 10 | 10 | 16 | 16 | 16 | 10 | 10 |
| 10 | 10 | 10 | 10 | 10 | 16 | 16 | 16 | 10 | 10 |
| 10 | 10 | 10 | 10 | 10 | 16 | 16 | 16 | 10 | 10 |
| 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |

图 6.24　保留了图像主要极大值的显示结果

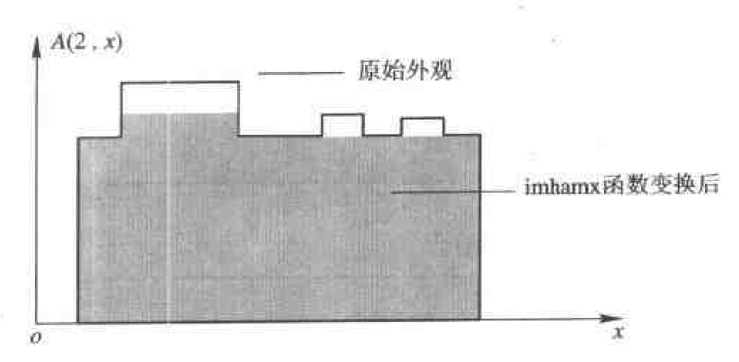



图 6.25　图 6.21 第二行的操作过程

可以使用 imimposemin 函数强调图像中指定的极小值。imimposemin 函数使用形态重构来消除图像中除指定极小值以外的其他所有的极小值，其调用格式如下：

```
I2 =imimposemin (I,BW,CONN)
```

其中，I 为输入图像，BW 是一个与 I 大小相同的二进制图像，其不为零的元素指定极小值，CONN 表示连通类型。假设需要对如图 6.26 所示的简单掩模图像中的两个最重要的

| 10 | 7 | 7 | 7 | 10 | 10 | 10 | 10 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 7 | 6 | 7 | 10 | 10 | 10 | 9 | 10 | 10 |
| 10 | 7 | 7 | 7 | 10 | 10 | 10 | 10 | 10 | 10 |
| 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| 10 | 10 | 10 | 10 | 10 | 2 | 2 | 2 | 10 | 10 |
| 10 | 10 | 10 | 10 | 10 | 2 | 2 | 2 | 10 | 10 |
| 10 | 10 | 9 | 10 | 10 | 2 | 2 | 2 | 10 | 10 |
| 10 | 9 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |

图 6.26　极小值待强调的掩模图像示意图

极小值进行强调，而删除其他极小值，则首先要创建一幅能够查明两个感兴趣的极小值的标记图像。可以通过明确地设置像素来指定极小值，或者使用其他形态函数来抽取掩模图像中希望强调的特征。以下代码使用 imextendedmin 函数来获得一幅指明两个极小值位置的二进制图像(参见图 6.27)：

```
marker = imextendedmin(mask,1)
```

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

图 6.27　指明两个极小值位置的二进制图像

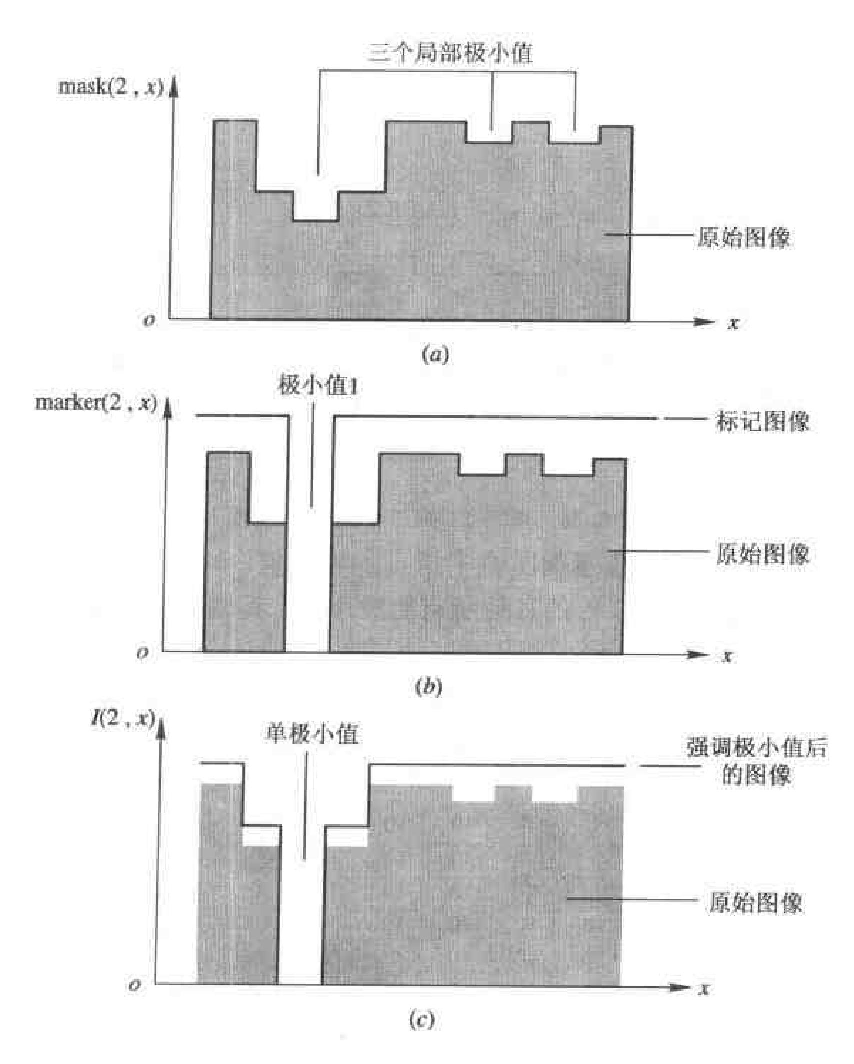



图 6.28　一维极小值强调过程示意图

(a) 一维数据；(b) 极小值 1 强调结果；(c) 其他极小值强调结果

使用 imimposemin 函数在掩模图像中有标记图像指定的点处创建新的极小值，读者可以通过观察结果来获得极小值的强调结果。注意，imimposemin 函数最终将标记图像指定的像素值设置为图像数据类型支持的极限值，imimposemin 函数还将修改图像中其他所有的像素值来删除其他的极小值。

```
I = imimposemin(mask,marker);
```

图 6.28 说明了 imimposemin 函数是如何修改图像第二行的外观的(即一维执行过程)。

# 6.4 二进制图像的形态学应用

## 6.4.1 距离变换

距离变换可以提供一个图像点距离估计矩阵，根据该矩阵可以进行图像分割。MATLAB 图像处理工具箱提供了函数 bwdist，该函数可计算二进制图像中每一个设置为 off(即数值为 0)的像素与其最近非零像素间的距离。bwdist 函数的调用格式如下：

```
[D,L] = bwdist (BW,METHOD)
```

其中，BW 为输入图像，METHOD 表示距离矩阵的类型，取值可以是'cityblock'、'chessboard'、'quasi-euclidean'或'euclidean'。bwdist 函数支持许多种距离矩阵，在缺省情况下计算的是欧氏距离。表 6.2 列出了这些距离矩阵的描述和例子。

**表 6.2 距离矩阵的类型和描述**

| 距离矩阵 | 描 述 | 图像及距离变换矩阵 | |
|---|---|---|---|
| 欧氏矩阵<br>'euclidean' | 欧氏距离是两个指定像素间的直线距离 | 0 0 0<br>0 1 0<br>0 0 0 | 1.41 1.0 1.41<br>1.0 0.0 1.0<br>1.41 1.0 1.41 |
| 城市矩阵<br>'cityblock' | 城市块距离矩阵根据4连通邻域估计像素间的路径。边缘相连的像素间隔1个单位，对角相连的像素间隔两个单位 | 0 0 0<br>0 1 0<br>0 0 0 | 2 1 2<br>1 0 1<br>2 1 2 |
| 棋盘矩阵<br>'chessboard' | 棋盘距离矩阵按照8连通邻域估计像素路径。边缘或角相连的像素间隔1个单位 | 0 0 0<br>0 1 0<br>0 0 0 | 1 1 1<br>1 1 1<br>1 1 1 |
| 准欧氏矩阵<br>'quasi-euclidean' | 准欧氏矩阵按照水平、垂直和对象线集合分段估计全部的欧氏距离 | 0 0 0 0 0<br>0 0 0 0 0<br>0 0 1 0 0<br>0 0 0 0 0<br>0 0 0 0 0 | 2.8 2.2 2.0 2.2 2.8<br>2.2 1.4 1.0 1.4 2.2<br>2.0 1.0 0 1.0 2.0<br>2.2 1.4 1.0 1.4 2.2<br>2.8 2.2 2.0 2.2 2.8 |

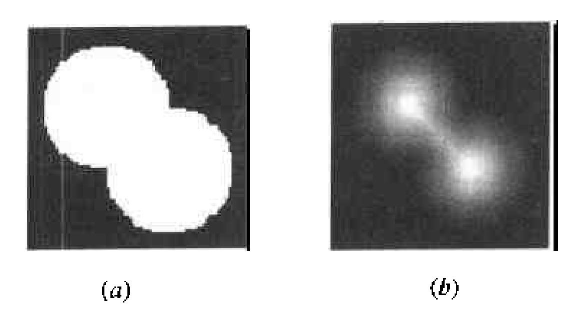
例如，以下代码将创建一幅包含两个相互交迭的圆形对象的二进制图像。

首先定义两个圆的圆心和半径。为了产生交叠，半径必须大于两圆心矩离的一半。

```
center1 = -10;
center2 = -center1;
dist = sqrt(2 * (2 * center1) ^ 2);
radius = dist/2 * 1.4;
lims = [floor(center1-1.2 * radius) ceil(center2+1.2 * radius)];
% 分别生成下面两个圆形对象的二进制图像
[x,y] = meshgrid(lims(1):lims(2));
bw1 = sqrt((x-center1).^ 2 + (y-center1).^ 2) <= radius;
bw2 = sqrt((x-center2).^ 2 + (y-center2).^ 2) <= radius;
bw = bw1 | bw2;
subplot(1,2,1), imshow(bw)
```

图像显示效果如图 6.29(*a*)所示。下面使用 bwdist 函数计算二进制图像的距离矩阵。注意，在距离变换后的图像(图 6.29(*b*))中，只有两个圆形的中心部分是白色的。

```
D = bwdist(bw);
subplot(1,2,2), imshow(D,[])
```



(*a*)　　(*b*)

图 6.29　距离变换前、后的显示效果比较

(*a*) 原图像；(*b*) 距离变换后

### 6.4.2　对象、区域和特征估计

MATLAB 图像处理工具箱提供许多利用形态操作计算二进制图像特征信息的函数，包括连接成分标记并使用标记矩阵获得图像的统计信息、选择图像中的对象、计算二进制图像中的面积、计算二进制图像的欧拉数等函数。下面我们将一一介绍这些函数的实现方法及用途。

#### 1. 连接成分标记

bwlabel 和 bwlabeln 函数可以实现连接成分标记。连接成分标记是一种用来辨识二进制图像中每一个对象的方法。这两个函数的调用格式分别如下：

```
L =bwlabeln (BW,N)
L = bwlabel (BW,CONN)
```

其中，BW 为输入图像，N 和 CONN 都表示连通类型，N 的取值可以是 4 或 8，缺省值为

8；CONN 的取值见表 6.1 说明。bwlabel 函数仅仅支持二维输入图像，而 bwlabeln 函数支持任意维数的输入图像，这两个函数都将返回一个称为标记矩阵的矩阵 L。标记矩阵是一幅与输入图像大小相同的图像，在该图像中不同的对象都使用不同的整数值来表示。例如，bwlabel 可以辨认以下二进制图像中的对象：

```
BW = [0 0 0 0 0 0 0 0;
      0 1 1 0 0 1 1 1;
      0 1 1 0 0 0 1 1;
      0 1 1 0 0 0 0 0;
      0 0 0 1 1 0 0 0;
      0 0 0 1 1 0 0 0;
      0 0 0 1 1 0 0 0;
      0 0 0 0 0 0 0 0];
X = bwlabel(BW,4)
X = 0 0 0 0 0 0 0 0
    0 1 1 0 0 3 3 3
    0 1 1 0 0 0 3 3
    0 1 1 0 0 0 0 0
    0 0 0 2 2 0 0 0
    0 0 0 2 2 0 0 0
    0 0 0 2 2 0 0 0
    0 0 0 0 0 0 0 0
```

在输出矩阵中，数值 1 代表第一个对象，2 代表第二个对象，以此类推(如果使用的是缺省的 8 连通邻域，那么只会存在数值 2，因为第一个和第二个对象对角相连，是同一个对象)。

bwlabel 和 bwlabeln 函数返回的标记矩阵都是双精度类型的，并不是一幅二进制图像。显示这个矩阵的一种方法就是使用 label2rgb 函数将其显示为一幅伪彩色索引图像。在伪彩色图像中，标记矩阵中辨识对象的数字将映射为相关图像调色板中的不同颜色。如果将标记矩阵显示为 RGB 图像，那么原图像中的对象将非常容易辨认。为了说明这个技术，本例使用 label2rgb 函数来显示标记矩阵 ***X***。label2rgb 函数的调用指定一个标准的 MATLAB 调色板 jet，label2rgb 的第三个参数 k 指定背景颜色为黑色。

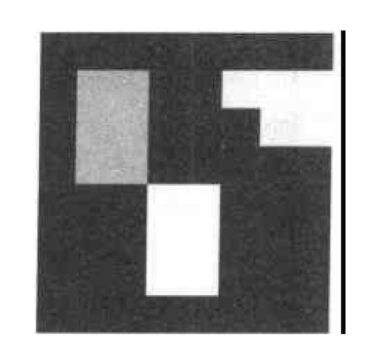


图 6.30 使用伪彩色显示标记矩阵

```
X = bwlabel(BW1,4);
RGB = label2rgb(X, @jet, 'k');
imshow(RGB,'notruesize')
```

显示结果如图 6.30 所示。

**2. 选择图像中的对象**

可以使用 bwselect 函数来选择二进制图像中的单个对象。通过指定输入图像的某一像素，bwselect 函数将返回一幅包含所有指定像素对象的二进制图像。可以使用交互或非交

互式的方法指定像素。如果使用以下调用格式，那么将进行非交互的像素指定：

```
BW2 = bwselect (BW1,C,R,N)
```

其中，BW 为输入图像，像素的坐标由(R,C)指定，如果R 和 C 是标量，那么将指定一个像素，否则指定一组像素。N 表示连通类型，取值为 4 或 8。

如果调用 bwselect 函数时没有指定任何输入参数，那么将采用交互式的像素选择方法。假设希望选择图像中显示在当前坐标轴上的对象，可输入命令：

```
BW2 = bwselect;
```

此时当位于图像中时光标将变为十字形，点击希望选择的对象，bwselect 函数将在用户所选择的每一个像素处显示一个小星形。所有选择都结束后点击【返回】命令，bwselect 就会返回一幅包含用户选择对象的二进制图像，同时删除所有星形。

### 3. 计算二进制图像的面积

bwarea 函数可以计算二进制图像的面积，其调用格式如下：

```
TOTAL = bwarea (BW)
```

面积是对图像前景大小的估计。粗略地说，面积就是图像中的像素数目，但是 bwarea 函数不仅仅是简单的输出设置为 on 的像素数目，而是在计算面积的过程中对每一个不同的像素模式加上不同的权值，这个加权过程是为了补偿将连续图像用离散的像素点描述时产生的误差。例如，一个包含 50 个像素的对角线比一个 50 像素的水平直线要长。作为 bwarea 函数的加权结果，水平线面积为 50，但是对角线面积为 62.5。以下代码将利用 bwarea 函数来判断因膨胀操作导致的图 6.31 所示图像面积增加的百分比。

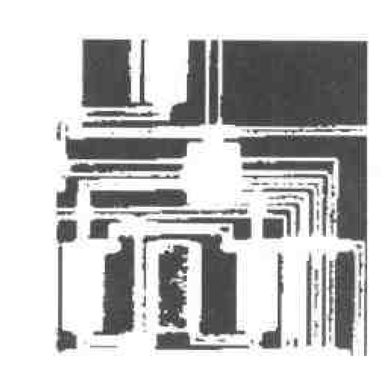


图 6.31　图像 circbw.tif 的显示效果

```
BW = imread('circbw.tif');
imshow(BW);
SE = ones(5);
BW2 = imdilate(BW,SE);
increase = (bwarea(BW2) - bwarea(BW))/bwarea(BW);
increase = 0.3456
```

计算图像中某个区域的面积以及该区域的周长，根据它们的比值分析该区域所代表的图像形状，这是一种常用的图像分析方法。

### 4. 计算二进制图像的欧拉数

bweuler 函数可以计算二进制图像的欧拉数，其调用格式如下：

```
EUL =bweuler (BW,N)
```

其中，N 表示连通类型。

欧拉数是对图像拓扑的估计，其定义为图像中的对象数目减去这些对象中孔洞的数目。在模式识别中，利用欧拉数进行聚类分析是一种很常用的有效方法。

bweuler 函数支持 4 连通或 8 连通邻域两种方式来进行欧拉数计算。以下代码使用 8 连通邻域计算图像 circbw. tif 的欧拉数。

```
BW1 = imread('circbw.tif');
eul = bweuler(BW1,8)
eul = -85
```

在这个例子中，欧拉数是个负数，表明洞的数目大于对象数目。

### 6.4.3 查表操作

在以上的几种操作中，某些对二进制图像的操作可以使用查表方法非常容易地实现。查表操作就是将经过某一函数邻域操作后像素所有可能的计算结果都记录下来，在进行其他像素处理时直接通过查表得到像素的取值，而不是重复进行计算。表通常是一个列向量，每一个元素代表要返回给一个可能的邻域像素联合的数值。可以使用 makelut 函数来针对不同的操作创建表，其调用格式如下：

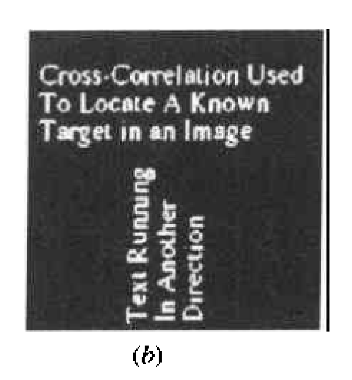
```
LUT = makelut (FUN,N,P1,P2,...)
```

其中，FUN 是一个返回标量的计算函数，N 表示邻域大小，P1、P2 等都是函数 FUN 的参数。makelut 函数可以按照 2×2(4 连通)或 3×3(8 连通)邻域来创建表。对于 2×2 邻域，有 16 个可能的邻域像素排列，因此创建的表是一个 16 元素向量；对于 3×3 邻域，因为有 512 种排列，所以创建的表是一个 512 元素的向量。

表一旦创建就可以通过调用 applylut 函数借助表来实现所需的操作。applylut 函数的调用格式非常简单，它以待处理图像和 makelut 函数创建所得的表 LUT 作为输入参数。




(a)




(b)

图 6.32　修改前、后图像的显示效果对比

(a) 修改前；(b) 查表操作修改后

**例 6.6**：使用查表操作修改图 6.32(a)所示图像所包含的文本。

首先需要编写一个计算函数，如果一个 3×3 邻域内有三个或三个以上像素数值为 1，那么该函数将返回 1，否则返回 0。然后调用 makelut，将写好的函数作为 makelut 的第一个参数，使用第二个参数来指定一个 3×3 的表，makelut 函数返回的 LUT 是一个数值为 1 或 0 的 512 元素向量，每一个数值都是函数相对于 512 种排列中的一种：

```
f = inline('sum(x(:)) >= 3');
lut = makelut(f,3);
```

最后使用 applylut 函数实现修改图像所含文本的操作：

```
BW1 = imread('text.tif');
BW2 = applylut(BW1,lut);
imshow(BW1)
figure, imshow(BW2)
```

结果如图 6.32(*b*)所示。

**注意：**

不能使用 makelut 和 applylut 函数对超过 2×2 或 3×3 的邻域进行操作，这些函数仅仅支持 2×2 或 3×3 邻域，这是因为对于大于 3×3 的邻域进行查表是不实际的。例如，针对 4×4 邻域的表将有 65536 个入口。

## 【习　题】

1. 调用 imfill 函数实现如图 6.19 所示的洞填充效果。

2. 以图 6.29 为例，比较该图像的四种距离变换(欧氏变换、城市矩阵等)的不同显示效果。

# 第七章 图像的空间变换

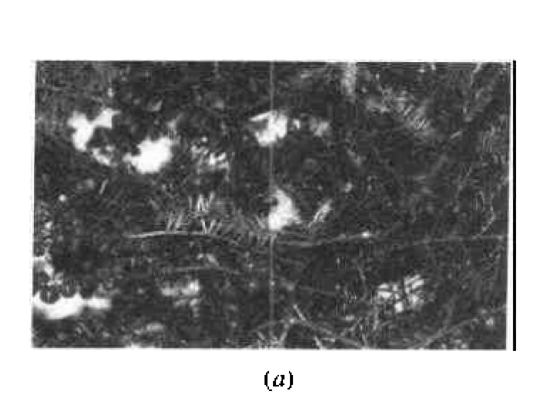
**本章要点：**

- ★ 空间变换
- ★ MATLAB 空间变换方法
- ★ MATLAB 的图像匹配
- ★ MATLAB 的图像投影

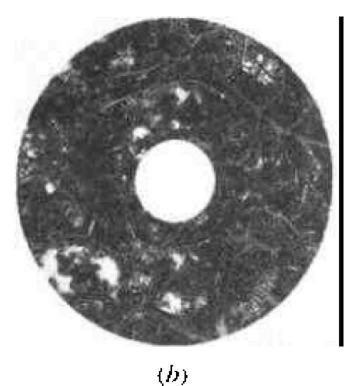
## 7.1 空间变换

### 7.1.1 空间变换

空间变换是一种非常有用的图像处理技术，在图像校正、图像匹配、图像变形等方面有着广泛的用途。例如，为了将一幅图像均匀地分布在一个圆环上，可以利用空间变换技术将如图 7.1(*a*)所示的图像变形为如图 7.1(*b*)所示的图像效果。



(*a*)



(*b*)

图 7.1 变形前、后的图像比较
(*a*) 变形前；(*b*) 变形后

在第三章中我们已经介绍了空间变换的概念和基本应用，本章将在此基础上对一些较为复杂的空间变换方法作出介绍。空间变换主要是用来保持图像中曲线的连续性和物体的连通性，一般都采用数学函数形式来描述输入、输出图像相应像素间的空间关系。空间变换的一般表达式为

$$g(x,y) = f(x',y') = f[a(x,y),b(x,y)] \tag{7.1}$$

其中，$g(x,y)$表示输出图像，坐标$(x',y')$指的是空间变换后的坐标，$a(x,y)$和 $b(x,y)$分

别表示图像在 x－y 坐标系中的空间变换函数。从式(7.1)可以看出，不同的 $a(x,y)$ 和 $b(x,y)$ 就定义了不同的空间变换。但是事实上，由于空间变换的复杂性和多样性，大多数空间变换都不是采用数学表达式来描述的，利用控制点进行变换是一种较为方便、实用的方法。图 7.2(*a*)和图 7.2(*b*)说明了使用控制点方法进行空间变换的基本原理，变换前、后的图像如图 7.2(*c*)和图 7.2(*d*)所示。使用的控制点即为图 7.2(*a*)和图 7.2(*b*)中的网格交叉点(图中用小圆点表示)。

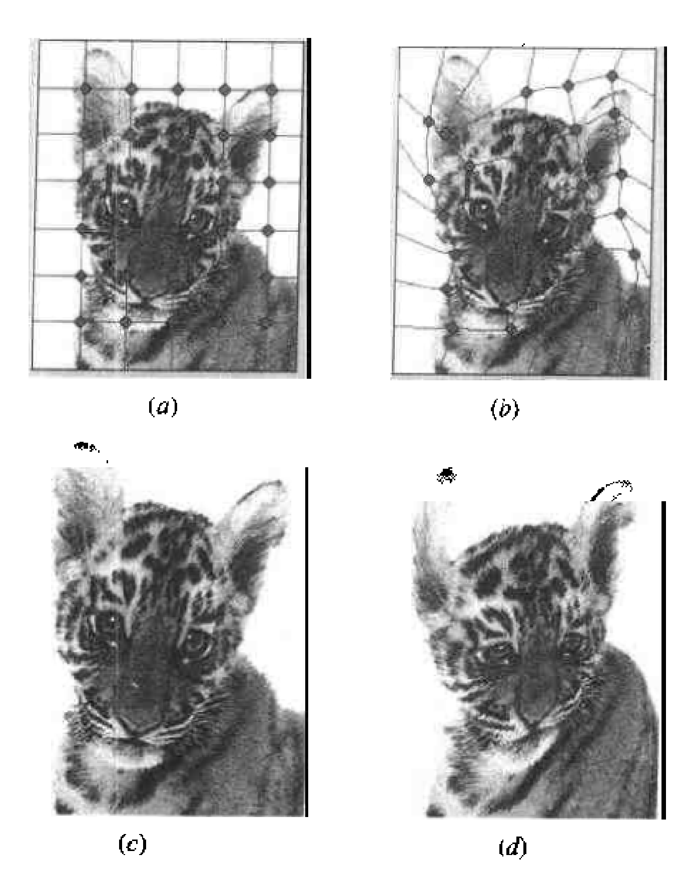


图 7.2　控制点变换原理及显示效果
(*a*) 原图像的网格；(*b*) 变换控制点；(*c*) 变形前的图像；(*d*) 变形后的图像

### 7.1.2　简单变换

简单变换是指坐标空间变换函数形式较为简单的变换，这种变换通常都用来实现图像的旋转、剪切等简单操作。例如，假设

$$\left.\begin{aligned} a(x,y) &= x + x_0 \\ b(x,y) &= y + y_0 \end{aligned}\right\} \tag{7.2}$$

那么就可以得到图像的平移运算：坐标为$(x_0, y_0)$的点将被平移为图像的原点，每一个图像点的移动距离为$\sqrt{x_0^2+y_0^2}$。如果令

$$\left.\begin{aligned} a(x,y) &= \frac{x}{t} \\ b(x,y) &= \frac{y}{t} \end{aligned}\right\} \tag{7.3}$$

那么图像就会在 $x$ 轴方向上放大 $s$ 倍，在 $y$ 轴方向上放大 $t$ 倍，此时原点保持不变。

图像旋转所采用的空间变换函数形式如下：

$$\left.\begin{aligned} a(x,y) &= x\cos\theta - y\sin\theta \\ b(x,y) &= x\sin\theta + y\cos\theta \end{aligned}\right\} \tag{7.4}$$

其中，$\theta$ 表示图像绕原点顺时针旋转的角度。显然，可以将各种不同的简单变换联合起来使用。例如，可以先对图像进行缩放再进行旋转。

有时采用分离操作的方法不但可以实现较为复杂的空间变换组合，还可以提高变换的执行效率。举例来说，图像进行平移放大的操作可以通过两个分离的步骤实现：首先在水平方向上进行平移放大，生成一幅中间图像，然后将中间图像作为输入，再在垂直方向上进行同样的运算就可以得到最终的结果。图像旋转操作就是一种可分离的操作：首先对图像进行水平变换，公式为

$$\left.\begin{aligned} a(x,y) &= x\cos\theta - y\sin\theta \\ b(x,y) &= y \end{aligned}\right\} \tag{7.5}$$

然后使用以下公式进行垂直变换：

$$\left.\begin{aligned} a(x,y) &= x \\ b(x,y) &= \frac{x\sin\theta + y}{\cos\theta} \end{aligned}\right\} \tag{7.6}$$

在分离操作的方式下，图像首先在水平方向上被压缩 $\cos\theta$ 倍，然后在垂直方向上进行扩展。将图像旋转的二维操作转换为一维运算来进行，这样能够简化操作，提高效率，但这种分离方法对旋转角度有一定的限制，当 $\theta=90°$ 时不能采用这种分离方法，此时可以通过简单的行、列互换来实现图像旋转。

### 7.1.3　控制点变换

对于相对简单的空间变换，使用解析表达式进行操作是切实可行的，然而，在许多图像处理应用中的空间变换都是相当复杂的，无法用简单的数学表达式表述。另外，大多数所需的空间变换需要从实际测量图像中获得，而不能根据函数描述来获得。为此，采用图像中一系列能够代表图像主要特征的控制点的位移值来描述空间变量是一种常用的空间变换方法。由于控制点只是图像中的一小部分像素点，所以其他像素的位移必须通过插值计算获得，其他像素使用的插值方法是控制点变换的关键所在，比较常用的方法有两种：多项式卷绕和图像分割卷绕变换。

多项式卷绕是利用控制点序列为 $a(x,y)$ 和 $b(x,y)$ 找到一个近似的多项式描述，该多项式的参数能够使多项式的取值与控制点及其位移量吻合。通常使用的多项式都是5阶以上的，如果多项式的项数与控制点数目相同，那么就可以设计出能准确映射控制点的空间变换形式，在这种情况下可以通过线性方程组求解得到多项式的系数。当控制点的数目多于多项式项数时，必须采用拟合方法来确定多项式系数。常用的数值拟合方法有奇异值分解、正交分解等，实践证明它们的拟合效果较好。

如果多项式卷绕难以实现，那么就应该将图像分割成很多矩形块进行卷绕变换，较常见的做法是将控制点形成一个矩形输入栅格，通过映射将栅格变换为输出图像中连通的水平输出栅格，控制点为每一个输出栅格对应的顶点，输入栅格内的各点映射为相应输出栅

格内的点。通常采用的栅格插值方法是双线性空间变换方法，其一般表达式如下：

$$G(x,y) = F(ax + by + cxy + d, e + fy + gxy + h) \tag{7.7}$$

由于四个控制点(输入栅格顶点)将映射为输出栅格的顶点，所以可以通过这一限制计算得出式(7.7)中的八个未知系数，然后对输入栅格内的每一个点使用式(7.7)进行计算，得出输出栅格的内点。由于双线性空间变换是一种能够保持连通性和连续性的光滑映射方法，所以通常能够取得很好的效果。

## 7.2 MATLAB空间变换方法

在第三章中我们已经介绍了调用函数 maketform 和 imtransform 进行空间变换的基本方法，本节将通过一个例子来说明这两个函数能够实现的各种空间变换类型。这里要预先介绍两个重要函数的使用方法：一个是多维数组几何变换函数 tformarray，另一个是创建重采样结构的 makeresampler 函数。

(1) makeresampler 函数的基本调用格式如下：

```
R =makeresampler (INTERPOLANT,PADMETHOD)
```

其中，INTERPOLANT 将指定所使用的插值核，取值可以是 'nearest'、'linear'或 'cubic'。PADMETHOD 指定边界的填充类型，取值可以是 'replicate'、'symmetric'、'circular'、'fill'或 'bound'。返回值 R 是一个 TFORM 结构，imtransform 函数将调用它来进行具体的图像变换。这种调用格式可以用来调用图像处理工具箱内置的空间变换类型，用户还可以通过以下调用格式自定义空间变换类型：

```
R = makeresampler (PropertyName,PropertyValue,...)
```

其中，PropertyName 可以包含以下几项内容：'Type'、'PadMethod'、'Interpolant'、'NDims'、'ResampleFcn'和'CustomData'。Type 是必须包含的一项内容，取值为'separable'或'custom'。如果 Type 的取值为'separable'，那么该调用结果与 makersampler 函数的第一种调用结果是相同的；如果取值为'custom'，那么除了最后一项内容以外，其他各项都是必需的。NDims 是一个正数，用来指定受控的采样方向，如果取值为 inf，则表示采样可以为任意方向。ResampleFcn 是采样函数的句柄，该函数的调用格式如下：

```
B = ResampleFcn(A,M,TDIMS_A,TDIMS_B,FSIZE_A,FSIZE_B,F,R)
```

其中，M 是将空间 A 映射到 B 所采用的具体映射方式。TDIMS_A 和 TDIMS_B 分别用来指定图像需要变换的维空间，FSIZE_A 和 FSIZE_B 分别指定 A 和 B 在指定维空间上的大小。

'CustomData'参数的取值与 ResampleFcn 的具体形式有关。

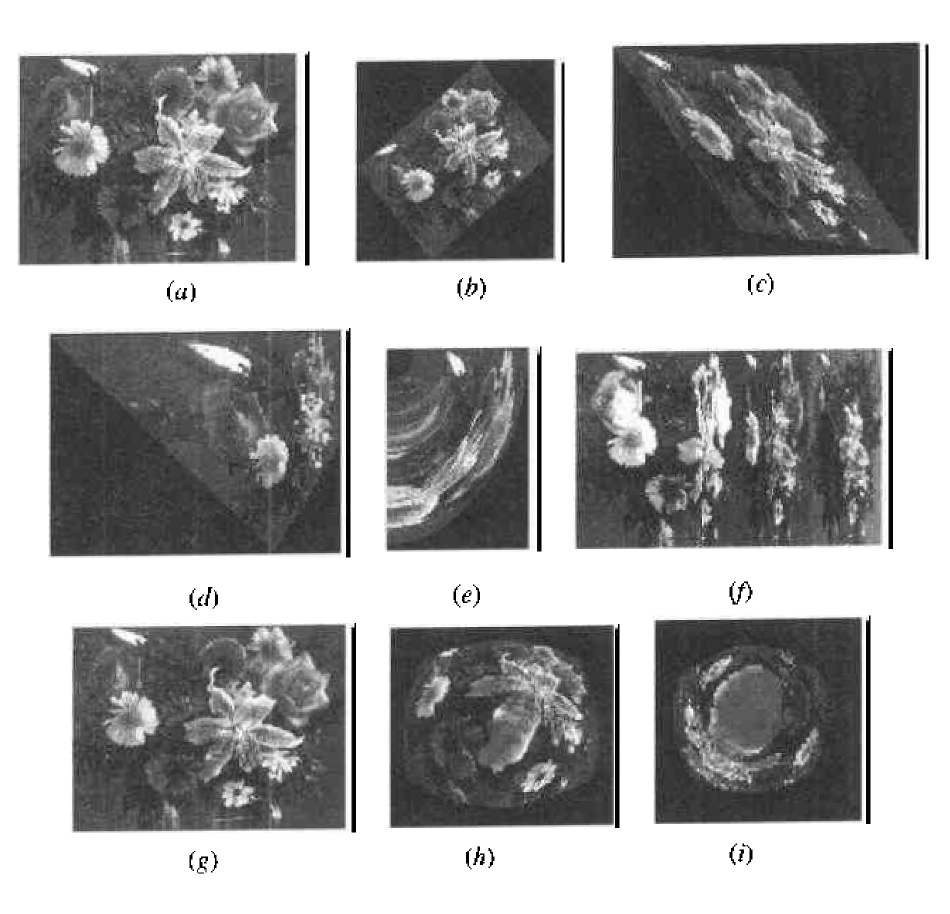
(2) tformarray 函数的基本调用格式如下：

```
B = tformarray(A,T,R,TDIMS_A,TDIMS_B,TSIZE_B,TMAP_B,F)
```

其中，T 是将图像 A 变换为图像 B 所使用的空间变换；R 是采样器，通常由 makeresampler 函数创建。TDIMS_A、TDIMS_B 和 TSIZE_B 的含义与采样函数的定义类似。TMAP_B 是一个可选参数，用来指定 B 元素与对应输出空间位置的关系，如果 TMAP_B 不为空，那么 TSIZE_B 必为空。F 是双精度填充数组。

**例 7.1**：用 maketform 和 imtransform 函数实现多种类型的空间变换。

通过设置 maketform 不同的变换类型参数(第一个参数)可以对图 7.3(*a*)所示的图像 I 进行以下几种空间变换，各种变换的结果如图 7.3(*b*)～(*i*)所示。



(*a*)　(*b*)　(*c*)

(*d*)　(*e*)　(*f*)

(*g*)　(*h*)　(*i*)

图 7.3　图像及其各种空间变换的显示效果

(*a*) 原始图像；(*b*) 线性投影；(*c*) 仿射变换；(*d*) 几何投影变换；(*e*) 多项式卷绕；(*f*) 分段线性变换；(*g*) 正弦变换；(*h*) 放射性变换；(*i*) 压缩性变换

1) 线性投影(旋转)变换

```
scale = 1.2;                %放大系数
angle = 40 * pi/180;        %旋转角度
tx = 0;                     %x 轴变换
ty = 0;                     %y 轴变换
sc = scale * cos(angle);
ss = scale * sin(angle);
T = [ sc -ss;
      ss sc;
      tx ty];
t_lc = maketform('affine',T);
I_linearconformal = imtransform(I,t_lc,'FillValues',.3);
subplot(332),imshow(I_linearconformal);
```

2) 仿射变换

```
T = [1 0.1;1 1;0 0];        %T 可以是不同大小的矩阵(下同)
```

```
t_aff = maketform('affine',T);
I_affine = imtransform(I,t_aff,'FillValues',.3);
subplot(333),imshow(I_affine)
```

3）几何投影变换

```
T = [1 0 0.008; 1 1 0.01;0 0 1];
t_proj = maketform('projective',T);
I_projective = imtransform(I,t_proj,'FillValues',.3);
subplot(334),imshow(I_projective)
```

4）多项式卷绕

```
xybase = reshape(randn(12,1),6,2);
t_poly = cp2tform(xybase,xybase,'polynomial',2);
T = [ 0 0; 1 0; 0 1;0.001 0; 0.02 0; 0.01 0];
t_poly.tdata = T;
I_polynomial = imtransform(I,t_poly,'FillValues',.3);
subplot(335),imshow(I_polynomial)
```

5）分段线性变换

```
imid = round(size(I,2)/2)I_left = I(:,1:imid);
stretch = 1.5;   % 缩放系数
size_right = [size(I,1) round(stretch * imid)];
I_right = I(:,imid+1:end);
I_right_stretched = imresize(I_right,size_right);
I_piecewiselinear = [I_left I_right_stretched];
subplot(336),imshow(I_piecewiselinear)
```

6）正弦变换

```
[nrows,ncols] = size(I);
[xi,yi] = meshgrid(1:ncols,1:nrows);
a1 = 5;   % 正弦函数幅值
a2 = 3;
u = xi + a1 * sin(pi * xi/imid);
v = yi - a2 * sin(pi * yi/imid);
tmap_B = cat(3,u,v);            %连接数组 u 和 v
resamp = makeresampler('linear','fill');
I_sinusoid = tformarray(I,[ ],resamp,[2 1],[1 2],[ ],tmap_B,.3);
subplot(337),imshow(I_sinusoid)
```

7）放射性变换(从图像中心向外放射性扩散)

```
xt = xi(:) - imid;             %变换弧度
yt = yi(:) - imid;
[theta,r] = cart2pol(xt,yt);
a = .001;                   % 放大系数
s = r + a * r.^ 3;
[ut,vt] = pol2cart(theta,s);
u = reshape(ut,size(xi)) + imid;
```

```
v = reshape(vt,size(yi)) + imid;
tmap_B = cat(3,u,v);
I_barrel = tformarray(I,[ ],resamp,[2 1],[1 2],[ ],tmap_B,.3);
subplot(338),imshow(I_barrel)
```

8) 压缩性变换(放射性变换的逆变换)

```
xt = xi(:) - imid;            % 变换弧度
yt = yi(:) - imid;
[theta,r] = cart2pol(xt,yt);
a = -.0005;                   % 放大系数
s = r + a*r.^3;
[ut,vt] = pol2cart(theta,s);
u = reshape(ut,size(xi)) + imid;
v = reshape(vt,size(yi)) + imid;
tmap_B = cat(3,u,v);
I_pin = tformarray(I,[ ],resamp,[2 1],[1 2],[ ],tmap_B,.3);
subplot(339),imshow(I_pin)
```

## 7.3　MATLAB的图像匹配

空间变换的一个主要应用就是对图像进行配准，使不同图像描述的同一物体的位置相同，以便进行进一步的图像比较，因而图像匹配通常是作为其他图像处理应用程序的预备步骤。例如，使用图像匹配技术将地球表面的卫星照片或不同的医学诊断特征(MRI或SPECT)图像匹配起来，然后通过比较配准图像的特征来观察江河的迁移、洪灾地区的情况，或者从MRI或SPECT图像中推断是否有肿瘤等。

图像匹配的主要目的是通过选择图像中的控制点(输入、输出图像的匹配位置)来推断空间变换的参数，然后对图像的其他像素应用推断出的空间变换，使得输出图像与给定基本图像所体现的场景一致(没有位移)，最后比较配准后的图像以获得真正有意义的图像差别。总体来说，图像匹配需要以下步骤：

(1) 将图像读入MATLAB工作平台中。

(2) 指定图像中的控制点对并保存。

(3) 使用互相关性进一步协调控制点对(此步骤可选)。

(4) 指定所需变换类型并根据控制点对推断变换参数。

(5) 变换未匹配的图像，将其插入图像队列中。

图7.4是图像匹配的流程图。

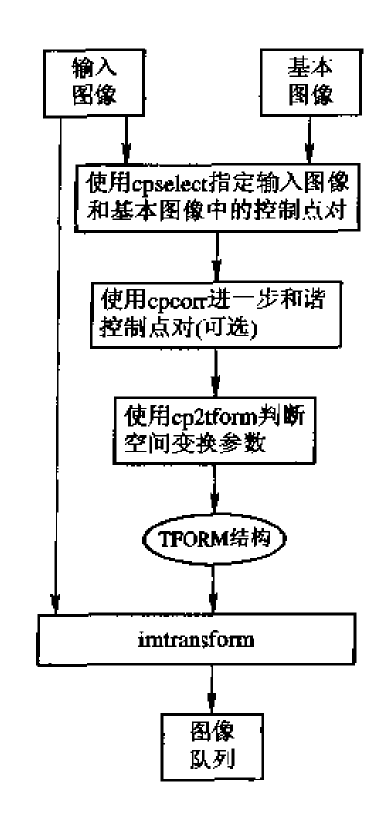



图7.4　图像匹配的流程图

MATLAB图像处理工具箱提供了一个控制点选择工具，使用户可以交互式地选择两幅图像中相应的控制点对。指定控制点分为以下四个步骤：

(1) 启动控制点选择工具，指定输入图像和基本图像。

(2) 观察图像，寻找在两幅图像中都可以识别的可视元素。cpselect 提供多种方法操纵图像，其中，平铺和缩放图像可获得图像观察区域内更多的细节信息。

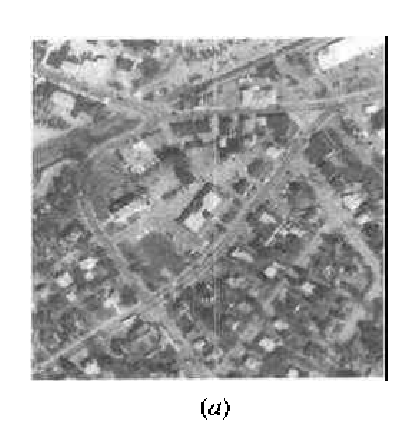
(3) 选定输入图像和基本图像中匹配的控制点对。

(4) 将控制点保存在 MATLAB 的工作平台中。

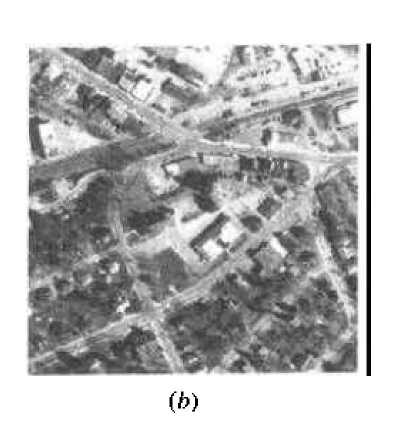
下面给出一个例子，说明如何通过选择控制点来实现图像匹配。

**例 7.2**：数字正色摄影图像匹配。

本例需要将一幅数字航空照片(图 7.5(*a*))匹配为一幅覆盖相同区域的数字正色摄影图像(图 7.5(*b*))。航空图像存在较为严重的几何失真，主要包含摄像机透视效果、地形和建筑模糊以及镜头失真等。正色摄影图像则是校正过的较为真实的场景描述。



(*a*)



(*b*)

图 7.5　数字航空照片与数字正色摄影图像的显示效果比较
(*a*) 数字航空照片；(*b*) 数字正色摄影图像

**步骤一**：将图像读入 MATLAB 中。

```
orthophoto = imread('westconcordorthophoto.png');          %读入标准图像
subplot(1,2,1), imshow(orthophoto)
unregistered = imread('westconcordaerial.png');            %读入待匹配图像
subplot(1,2,2), imshow(unregistered)
```

虽然在显示图像时并不一定要将图像读入 MATLAB 中，但是如果希望使用互相关性调节控制点的位置，那么图像必需位于 MATLAB 工作平台中。

**步骤二**：选择图像中的控制点。

由于未匹配的图像是一幅 RGB 图像，而控制点选择工具只能接受灰度图像，所以本例仅仅将 RGB 图像的一个颜色分量传递给 cpselect。首先在 MATLAB 提示符后键入以下语句：

```
cpselect(unregistered(:,:,1),orthophoto)
```

cpselect 函数显示输入图像和基本图像两个视图，用户可以通过点击鼠标来选择控制点。图 7.6 显示了选择四对控制点后的控制点选择工具的外观。用户选择的控制点对数目与需要实现的变换类型密切相关。这里要注意的是，控制点选择工具同时仅支持一个颜色通道，所以用户看到的图像将是彩色图像的某一个颜色分量图像，但是最终的结果必须应用于图像的三个颜色分量。

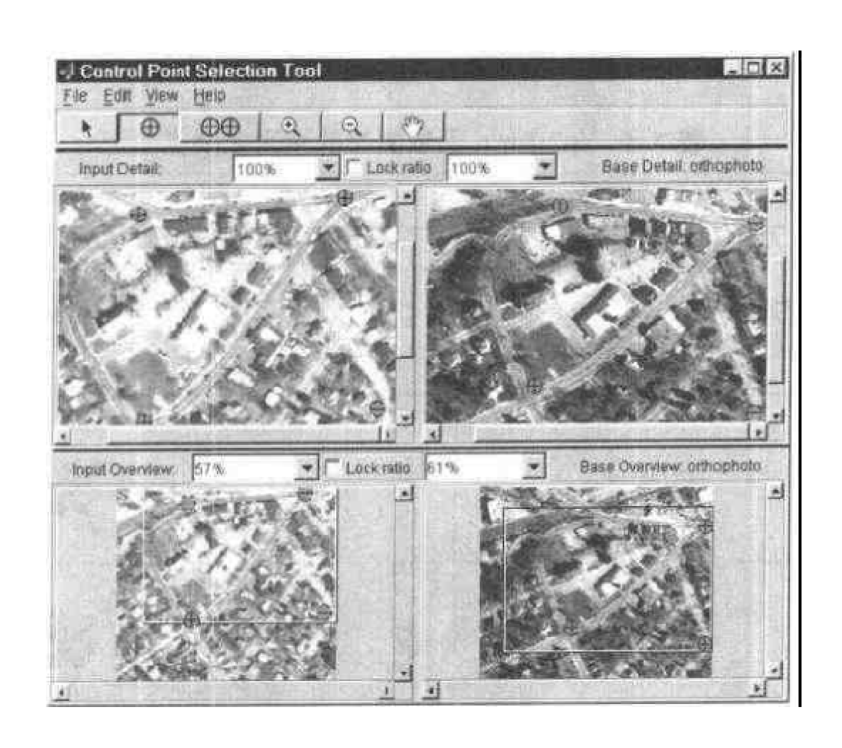



图 7.6　控制点选择工具界面

**步骤三：**将控制点对保存在 MATLAB 工作平台中。

在控制点选择工具中，选择【File】菜单的【Save Points to Workspace】选项就可以将所选的控制点保存在一个变量中，同时返回到原界面。本例中假设所选的输入图像控制点集合如下，这些数值代表点的空间坐标，左边的列代表 x 坐标，右边的列代表 y 坐标：

```
input_points =120.7086   93.9772
              319.2222   78.9202
              127.9838   291.6312
              352.0729   281.1445
```

**步骤四：**进一步调节控制点对的放置。

这一步骤将使用互相关性来调节控制点选择工具选择的控制点位置，这一步不是必需的。注意，cpcorr 函数只能调节具有相同方向和大小的图像点对，不能做相对旋转，因此本例中的 cpcorr 函数实际上是不能调节所选控制点的。如果控制点无法调节，则 cpcorr 函数将返回未修改的控制点。

```
input_points_corr = cpcorr(input_points,base_points,...
                    unregistered(:,:,1),orthophoto)
input_points_corr =120.7086   93.9772
                   319.2222   78.9202
                   127.1046   289.8935
                   352.0729   281.1445
```

**步骤五：**指定变换类型并推断参数。

在这个步骤中将控制点传递给 cp2tform 函数，由该函数来判断图像匹配的变换参数。cp2tform 函数是一个数据拟和函数，可以根据控制点对的几何关系判断变换参数。

cp2tform函数返回一个几何变换结构的参数(TFORM 结构)。调用 cp2tform 时必须选择一种能够正确消除输入图像失真的变换类型。表 7.1 列出了 cp2tform 函数支持的六种空间变换类型。

**表 7.1　cp2tform 函数支持的空间变换类型**

| 变换类型 | 函数参数 | 最小控制点对 | 图例 |
|---|---|---|---|
| 线性等角变换 | 'linear conformal' | 2 对 | |
| 仿射变换 | 'affine' | 3 对 | |
| 投影变换 | 'projective' | 4 对 | |
| 多项式变换 | 'polynomial' | 二维：6 对；三维：10 对；四维：16 对 | |
| 分段线性变换 | 'piecewise linear' | 4 对 | |
| 局部权平均变换 | 'lwm' | 6 对(建议 12 对) | |

表中的前四个变换(线性变换、仿射变换、投影变换和多项式变换)都是全局变换，这些变换对整幅图像应用同一个数学表达式；后两种变换(分段线性变换和局部权平均变换)都是局部变换，这两种变换将对图像的不同部分使用不同的数学表达式。如果希望观察不同的变换将对操作的图像产生怎样的效果，首先试一下全局变换，如果这些变换得不到满意的效果再使用局部变换，此时先使用一下分段线性变换，然后再使用局部权平均变换。

上例中的航空图像失真的主要原因是摄像机的透视效果，忽视地域模糊(在本例中地域模糊影响较小)时可以使用投影变换来纠正这种失真。投影变换还将进行图像旋转，使之符合基本图像的地图坐标系统；其他失真效果可以通过给出地域和摄像机的足够信息，使用 maketform 创建一个合成变换来同步消除。

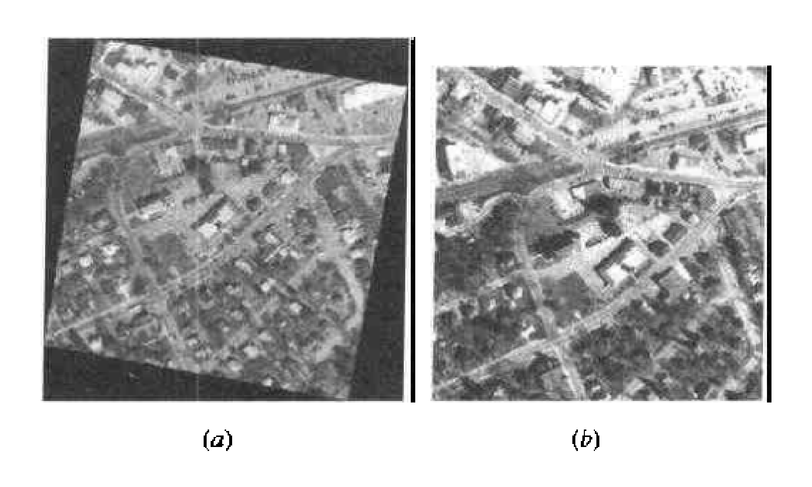
```
mytform = cp2tform(input_points,base_points,'projective');
```

**步骤六**：变换未匹配的图像。

这是图像匹配的最后一个步骤。以输入图形和定义变换的 TFORM 结构为参数调用 imtransform 函数，实现待匹配图像的空间变换，imtransform 函数将返回变换后的图像：

```
registered = imtransform(unregistered,mytform)
```

该变换将应用于输入图像的每一个颜色分量。比较变换后的图像和基本图像(如图 7.7(*a*)、(*b*)所示)可以观察到，经过图像匹配后，航空照片与正色摄影图像中相同坐标处的像素(除黑色边界以外)是一一对应的。



(*a*)　　(*b*)

图 7.7　匹配后的航空照片与正色摄影图像的显示效果比较

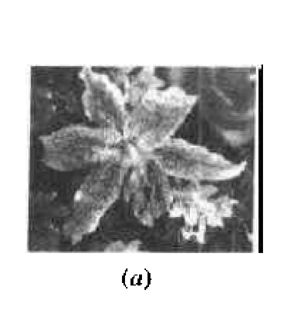
(*a*) 匹配后的航空照片；(*b*) 正色摄影图像

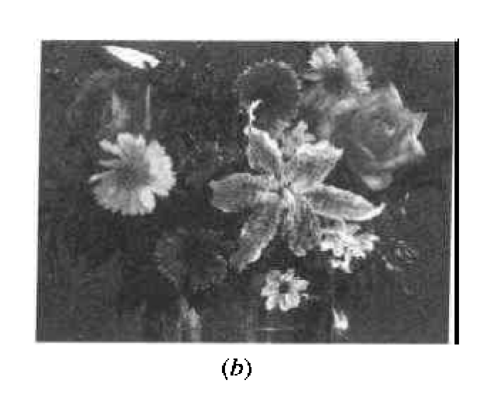
除了使用控制点判断空间变换类型从而进行图像匹配以外，利用相关性分析也可以实现图像匹配。下面给出一个例子说明相关性分析在图像匹配中的应用。

**例 7.3**：对图 7.8 所示的两幅图像进行匹配。

**步骤一**：读取图像。

```
lily = imread('lily.tif');
flowers = imread('flowers.tif');
subplot(1,2,1),imshow(lily);
subplot(1,2,2),imshow(flowers);
```





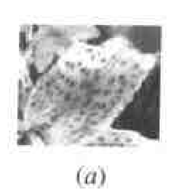
(*a*)　　(*b*)

图 7.8　需要匹配的两幅图像

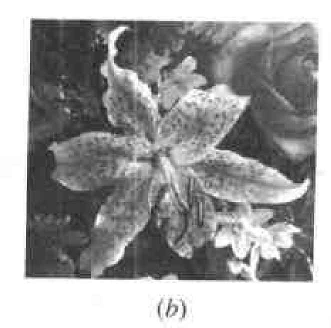
(*a*) 图像 lily.tif；(*b*) 图像 flowers.tif

**步骤二**：选择每一幅图像的子区域。在这一步骤中，用户可以通过鼠标选择需要截取的图像部分，此处假设截取结果如图 7.9 所示。

```
rect_lily = [93 13 81 69];
rect_flowers = [190 68 235 210];
sub_lily = imcrop(lily,rect_lily);
sub_flowers = imcrop(flowers,rect_flowers);
[sub_lily,rect_lily] = imcrop(lily);
[sub_flowers,rect_flowers] = imcrop(flowers);
subplot(1,2,1), imshow(sub_lily)
subplot(1,2,2), imshow(sub_flowers)
```



(a)



(b)

图 7.9 选择的子图像
(a) 子图像 1；(b) 子图像 2

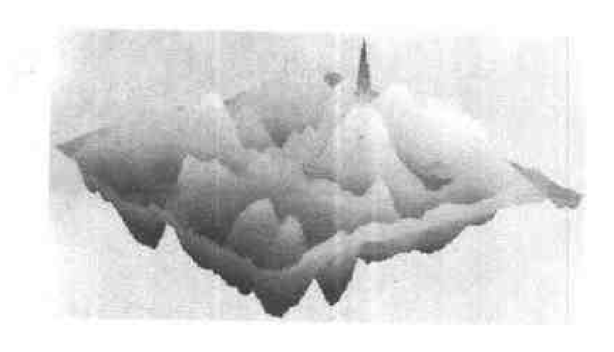


图 7.10 两幅子图像的相关性图形

**步骤三**：使用相关性分析两幅子图像。(得到的相关性图形如图 7.10 所示。)

```
c = normxcorr2(sub_lily(:,:,1),sub_flowers(:,:,1));
figure, surf(c), shading flat
```

**步骤四**：找到整幅图像的偏移。

```
[max_c, imax] = max(abs(c(:)));
[ypeak, xpeak] = ind2sub(size(c),imax(1));
corr_offset = [(xpeak-size(sub_lily,2))
(ypeak-size(sub_lily,1))];
rect_offset = [(rect_flowers(1)-rect_lily(1))
              (rect_flowers(2)-rect_lily(2))];
offset = corr_offset + rect_offset;
xoffset = offset(1);
yoffset = offset(2);
```

**步骤五**：观察待匹配图像是否位于基本图像中。

```
xbegin = xoffset+1;
xend = xoffset+ size(lily,2);
ybegin = yoffset+1;
yend = yoffset+size(lily,1);
extracted_lily = flowers(ybegin:yend,xbegin:xend,:);
if isequal(lily,extracted_lily)
   disp('lily.tif was extracted from flowers.tif')
end
```

**步骤六**：填充待匹配图像并将背景置为透明，然后贴到基本图像上以体现配准效果。

```
recovered_lily = uint8(zeros(size(flowers)));
recovered_lily(ybegin:yend,xbegin:xend,:) = lily;
[m,n,p] = size(flowers);
mask = ones(m,n);
i = find(recovered_lily(:,:,1)==0);
mask(i) = .2;          % 可以定义不同的透明度，此处选择为 0.2
figure, imshow(flowers(:,:,1))   %仅显示基本图像的红色分量
hold on
h = imshow(recovered_lily);   % 粘贴匹配图像
set(h,'AlphaData',mask)
```

最终得到的匹配图像如图 7.11 所示。

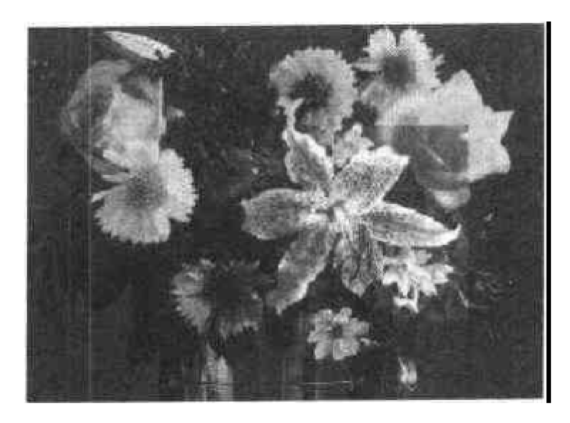


图 7.11　图像匹配效果

# 7.4　MATLAB 的图像投影

空间变换的另一个主要应用是地图绘制中的图像投影。例如，根据卫星传回来的图像，利用空间变换技术拼成地球、月球以及行星照片。输入图像和投影图像之间的空间变换的主要目的是确定输出图像的点与拍摄物体表面上点的对应关系。一般的图像投影技术都是在输出图像上覆盖一个矩形控制网格，然后根据图像点与物体表面点的关系将输出图像映射为一幅输入图像，从而得到较为真实的场景描述。

这里需要说明三个重要的地图性质：如果地图沿某些直线的尺度保持不变，那么称该地图具有等距性；如果地图中区域的面积保持不变，那么称该地图具有等值性；如果投影中的角度保持不变，那么称该地图具有保角性(也称为同形性)。不同的投影技术能够产生具有不同性质的地图。由于一幅具有保角性的地图在一点处保持形状不变将意味着物体细小特征只有轻微的畸变，所以通常采用的投影技术都将产生保角地图。对保角图像来说，如果给定一个解析变换函数 $F(u,v)=(x,y)$，那么必有 $x+\mathrm{i}y=F(u+\mathrm{i}v)$。

常用的保角变换有：球极平面投影法、Mercator 投影法和 Lambert 保角圆锥投影法，这三种投影方法的基本原理如图 7.12 所示，其中，球极平面投影法将球表面特征映射到一个平面(镜面)上；Mercator 投影法将球表面特征映射到一个圆柱面上；而 Lambert 保角圆

锥投影法将球表面特征映射到一个圆锥面上。事实上，图像的投影除了在地图映射方面非常有用之外，对于一般的图像而言，图像投影能够产生一种特殊的空间效果。下面通过一个例子来说明 MATLAB 的投影方法，读者可以从中体会到图像投影方法的精髓，以便将其进一步应用于其他图像处理领域中。

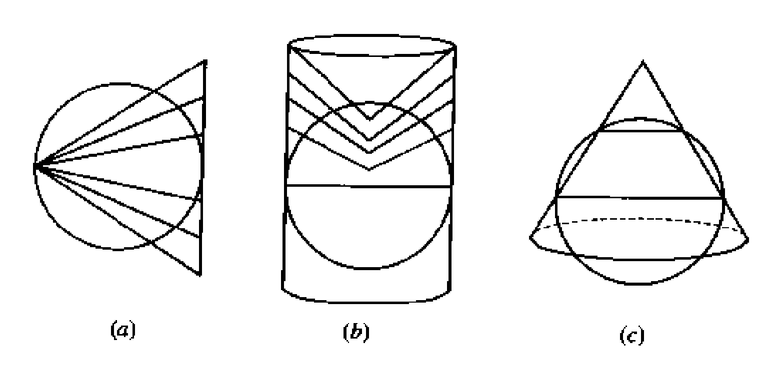



图 7.12　投影方法基本原理示意图
($a$) 球极平面投影；($b$) Mercator 投影；($c$) Lambert 保角圆锥投影

**例 7.4**：将如图 7.13($a$)所示图像进行保角变换，使之变为如图 7.13($b$)所示的外观。

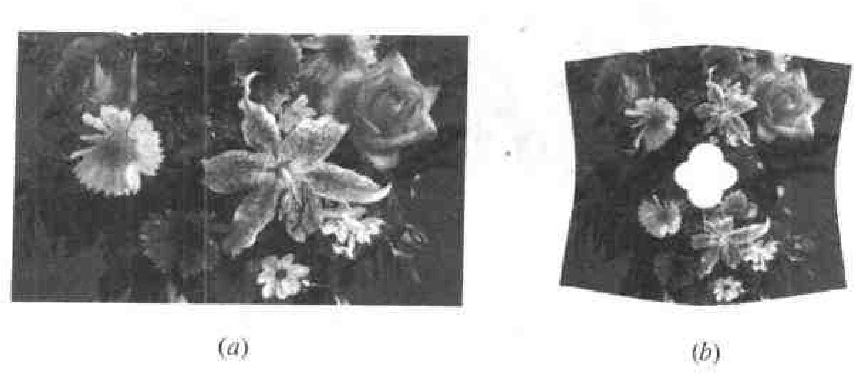



图 7.13　保角变换图像示例
($a$) 原图像；($b$) 保角变换后

上例中采用的空间变换函数为 $w=(z+1/z)/2$，其中 $z$ 和 $w$ 都是复数，分别表示输入和输出图像的坐标，即 $z=x+\mathrm{i}y$，$w=u+\mathrm{i}v$。这个保角变换能够将一个圆圈变换为一条直线。

**步骤一**：读取图像。

```
A = imread('flowers.tif');
A = A(31:330,1:500,:);
subplot(1,2,1),imshow(A)
```

**步骤二**：创建一个自定义的保角变换，然后调用 maketform 函数。

```
function U = ipex004( X, t )
Z = complex(X(:,1),X(:,2));
R = abs(Z);
W = (Z + 1./Z)/2;
U(:,2) = imag(W);
```

```
U(:,1) = real(W);
conformal = maketform('custom', 2, 2, [ ], @ipex004, [ ]);
```

**步骤三**：定义输入、输出图像的复平面大小。

```
uData = [ -1.25      1.25];     % 输出图像实部范围
vData = [  0.75     -0.75];     % 输出图像虚部范围
xData = [ -2.4       2.4 ];     % 输入图像实部范围
yData = [  2.0      -2.0 ];     % 输入图像虚部范围
```

**步骤四**：图像变换。

```
B = imtransform( A, conformal, 'cubic', ...
                'UData', uData,'VData', vData,...
                'XData', xData,'YData', yData,...
                'Size', [300 360], 'FillValues', 255 );
subplot(1,2,2), imshow(B);
```

变换后的图像是由原始图像的两次变换构成的，除了边界变为弧线以外，原始图像的其他特征都被保留下来了。

## 【习　　题】

1. 利用保角变换实现如图 7.1 所示的变换。
2. 利用控制点选择工具实现如图 7.2 所示的变换。

# 第八章 图像增强

**本章要点：**

★ 灰度变换增强

★ 空域滤波增强

★ 频域增强

★ 色彩增强

## 8.1　灰度变换增强

### 8.1.1　图像增强技术分类

图像增强的主要目的是改善图像的质量。对于一幅给定的图像，图像增强可以根据图像的模糊情况和应用场合，采用某种特殊的技术来突出图像中的某些信息，削弱或消除某些无关信息，从而有目的地强调图像的整体或局部特征。增强后的图像往往能够增强对特殊信息的识别能力，常常用来改善人对图像的视觉效果，让观察者能够看到更加直接、清晰、适于分析的信息。机器识别中的图像增强预处理往往直接影响着机器的感知和理解能力。

常用的图像增强技术有图像数据统计分析处理、直方图修改、图像平滑滤波、图像锐化等，这些技术可以单独使用，也可以联合应用。应该指出的是，图像增强没有固定不变的理论方法，增强质量主要是人根据增强目的而由主观视觉评定的，因而一般在得到满意的结果之前都会进行多次反复的试验和修改。

图像增强技术从总体上来说可以分为两个大类：频域增强方法和空域增强方法。空域增强方法是直接对图像中的像素进行处理，从根本上说是以图像的灰度映射变换为基础的，所用的映射变换类型取决于增强的目的。空域增强方法大致分为三种，它们分别是用于扩展对比度的灰度变换、清除噪声的各种平滑方法和增强边缘的各种锐化技术。灰度变换主要是利用点运算来修改图像像素的灰度，是一种基于图像变换的操作；而平滑和锐化都是利用模板来修改像素灰度，是基于图像滤波的操作。

为了有效和快速地对图像进行处理和分析，常常需要将原定义在图像空间中的图像以某种形式转换到其他空间中，然后利用该空间的特有性质方便地进行图像处理，最后再转换回原图像空间中，从而得到处理后的图像。最常用的变换空间就是频域空间。频域增强方法的关键在于图像的空域与频域变换类型。

以上两种增强方法是根据图像处理的空间不同而进行划分的，事实上图像增强技术的

种类还有很多。例如，如果按照图像处理策略的不同进行划分，可以将增强技术分为全局处理和局部处理两种；如果按照处理图像种类的不同来划分，可以分为灰度图像处理和彩色图像处理两种。读者可以根据自身的需要来选择合适的增强方法。本节的主要内容是介绍空域中的灰度变换增强技术，在以后的各小节中将分别介绍空域滤波增强技术、频域增强技术和彩色增强技术，其中，前面三种技术主要是针对灰度图像而言的，最后一种技术则是将图像增强技术扩展到彩色图像中去。

## 8.1.2　像素值及其统计特性

图像的灰度变换方法有很多种，其基本原则是利用某种变换函数对图像进行点运算，从而修改图像的像素灰度值。显然，为了选择一种合理的变换函数，首先就要对原始图像的像素灰度值有个大概的了解，然后根据像素的统计特性来确定需要的变换函数类型。直方图是灰度变换技术中最常用的像素统计特性描述方式，除此之外还有单个点的像素值、某一线段上的像素灰度分布、图像的等高线图、图像的统计摘要(包括均值、方差等)以及区域特性度量等方式。MATLAB 图像处理工具箱提供了许多返回图像数据矩阵统计信息的函数。下面我们一一介绍这些函数的功能和用法。

### 1. 单个像素的选择

MATLAB 图像处理工具箱包括两个能够提供指定像素信息的函数：pixval 函数和 impixel 函数。当鼠标在图像上移动时，pixval 函数将交互式地显示像素的数据值，另外它还可以显示两个像素间的欧几里得距离；impixel 函数返回被选择像素或像素集合的数据值，既可以通过使用输入参数定义像素坐标，又可以使用鼠标来选择。

使用 pixval 函数必须首先显示图像，然后输入 pixval on 命令打开图像窗口进行交互访问。pixval 将在图形窗口的底部自动添加一个黑色的状态栏，这个状态栏将显示当前鼠标所在像素的 x 和 y 坐标以及该像素的颜色数据。如果用户在图像中点击并拖动鼠标，那么 pixval 还将显示最初点击点与鼠标当前所在像素之间的欧几里得距离。如果想退出交互操作，可输入 pixval off 命令。如果按照以下格式调用 pixval，那么将打开由参数 FIG 指定的窗口进行交互访问(OPTION 参数可以为 on 或 off)。

```
pixval(FIG,OPTION)
```

pixval 函数给出的像素灰度信息比函数 impixel 多，但是 impixel 函数的优势在于它能够将结果返回到一个变量中，以后可以通过交互式或非交互式的方法对这个变量进行访问或操作。交互式 impixel 函数的调用格式如下：

```
[C,R,P]=impixel(X,MAP)
```

其中，X 表示输入图像，MAP 为索引图像的调色板(仅当图像为索引图像时才有此参数)，C 表示指定像素的颜色，R 和 P 表示像素的坐标。如果在输入图像参数后面给出两个指定像素坐标的向量，那么 impixel 函数将返回指定像素的灰度；如果调用 impixel 函数时没有指定输入参数，那么系统将自动选择位于当前坐标轴中的图像。在交互方式下，选择完毕后点击【返回】命令，impixel 函数将返回被选像素的颜色数据。例如，以下代码首先调用 impixel 函数，通过交互方式选择三个像素，点击【返回】命令后得到所选像素的数值：

```
imshow canoe.tif
vals = impixel
```

三个被选像素的颜色数据如下：

```
vals =
        0.1294 0.1294 0.1294
        0.5176 0 0
        0.7765 0.6118 0.4196
```

从图 8.1 中可以看出，第二个像素位于图中的小船上，颜色应该为纯红色，与输出的数据一致。



图 8.1　获取单个像素数值示意图

⚠ 说明：

对于索引图像，pixval 和 impixel 函数都将显示调色板存储的 RGB 值，而不是调色板的索引。

⚠

**2. 线段上像素灰度分布的计算和绘制**

函数 improfile 能够计算并绘制图像中一条或多条线段上的所有像素的灰度值。调用该函数时，可以使用端点坐标作为输入参数来定义线段，也可以使用鼠标交互式地定义线段。非交互式 improfile 函数的调用格式如下：

```
C = improfile(I,xi,yi)
```

其中，I 为输入图像，xi 和 yi 是两个向量，用来指定线段的端点。C 是线段上各点的灰度或颜色值。如果调用 improfile 函数时不指定任何输入参数，那么当鼠标位于图像中时会变为十字形，可以通过点击鼠标定义线段的端点。点击【返回】命令后，improfile 函数将在一个新的图形窗口中显示所得的线段灰度值的分布情况。

无论是交互还是非交互模式，improfile 函数都将使用插值方法来确定曲线上等间隔点的数值(缺省情况下，improfile 函数使用最近邻域插值方法，也可以使用自定义的插值方法)。improfile 函数在处理灰度图像和 RGB 图像时能够获得非常好的效果。对于单独的线段，improfile 函数将在二维视图中绘制点的灰度值；对于多条曲线，improfile 函数将在三维视图中绘制灰度值。下例首先调用 improfile 函数，然后使用鼠标指定一条线段，该线段(图 8.2(a)中的灰色线段表示)的起点在左边，得到的像素分布结果如图 8.2(b)所示。

```
imshow ('debye1.tif')
improfile
```

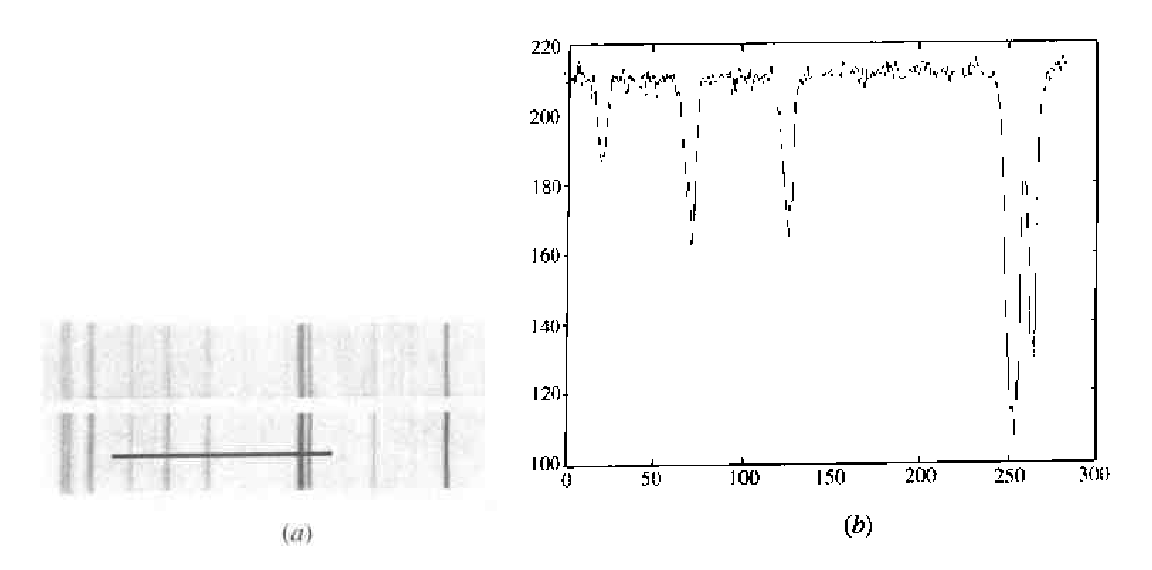



图 8.2　线段上的像素灰度分布示意图

(a) 原图像；(b) 像素灰度分布曲线

对于 RGB 图像，improfile 函数将显示所选线段上像素的红、绿、蓝颜色分量的数值。下例说明了 improfile 函数如何操作 RGB 图像。图 8.3(a)中的白线表明选择的线段，起点在上方，灰度分布曲线如图 8.3(b)所示。线段与图像颜色具有一一对应的关系，例如，图形的中心区域说明了红色和绿色具有高亮度，而蓝色亮度为 0，这些数值对应于图像中的黄色花朵。

```
imshow ('flowers.tif')
improfile
```

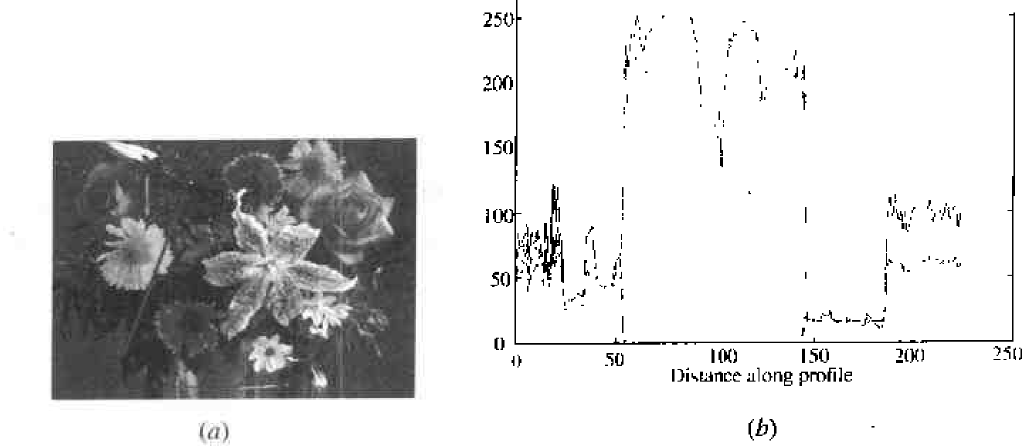


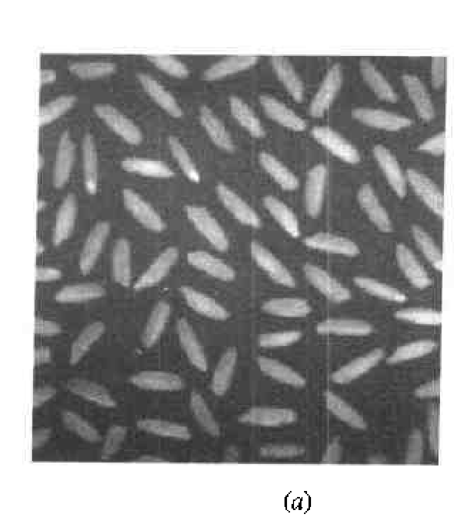

图 8.3　RGB 图像线段灰度分布示意图

(a) 原图像；(b) 像素灰度分布曲线

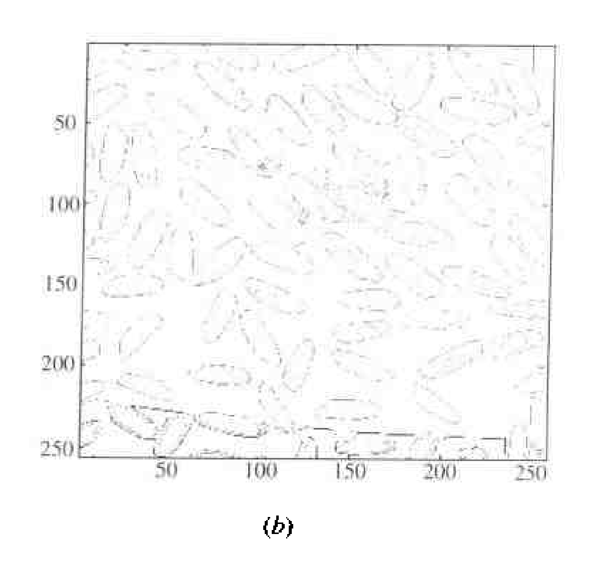
### 3. 图像等高线

可以使用工具箱函数 imcontour 来显示灰度图像数据的等高线图形，这个函数与 MATLAB 的 contour 函数类似，但是 imcontour 函数能够自动进行坐标轴设置，使输出图形的方向和外观与图像吻合。imcontour 函数的调用格式非常简单，就是以图像为输入参数，输出结果为图像的等高线图形。下例将显示一幅灰度图像(如图 8.4(*a*)所示)以及图像数据的等高线图(如图 8.4(*b*)所示)：

```
I = imread('rice.tif');
imshow(I)
figure, imcontour(I)
```



(*a*)




(*b*)

图 8.4　灰度图像与其等高线图形比较

(*a*) 灰度图像；(*b*) 等高线图形

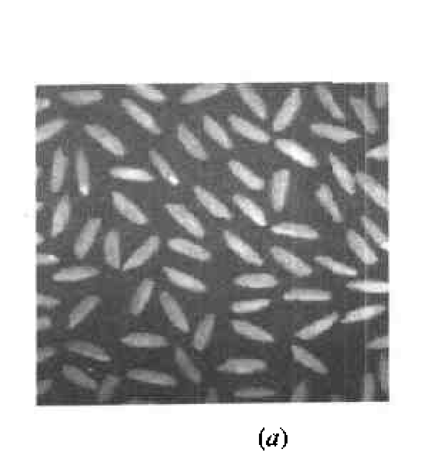
对于 imcontour 函数生成的图形，可以使用 MATLAB 的标记函数对其进行标记。例如，使用 clabel 函数来生成一个坐标轴标签。

### 4. 直方图

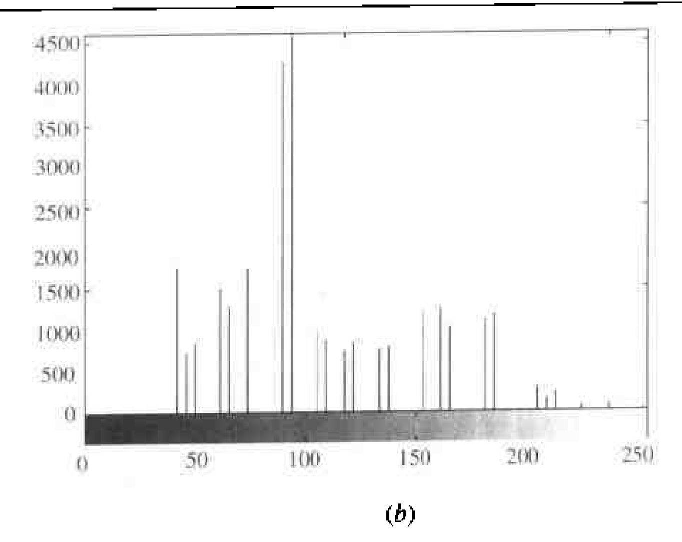
图像的直方图是一个显示灰度或索引图像亮度分布情况的图表。图像直方图函数 imhist 通过使用 n 个等间隔的柱(每一柱代表一个数值范围)来创建这个图表，然后计算每个范围内的像素个数。imhist 函数的调用格式很简单，它以图像和所需的柱数目作为输入参数，自动绘制图像的直方图。例如，以下命令将显示一幅米粒图像(如图 8.5(*a*)所示)和一个 64 柱的直方图(如图 8.5(*b*)所示)。

```
I = imread('rice.tif');
imshow(I)
figure, imhist(I,64)
```

根据直方图显示，数值 100 附近出现了一个高峰，这个高峰对应于米粒图像中的背景像素。



(a)




(b)

图 8.5　图像及其直方图比较
(a) 原图像；(b) 图像的直方图

**5. 统计摘要**

可以使用工具箱统计函数 mean2、std2 和 corr2 来计算图像的标准统计特性。mean2 和 std2 函数计算矩阵元素的平均值和标准偏差；corr2 函数计算两个相同大小矩阵的相关系数。事实上，以上这些函数都是 MATLAB 内核函数 mean、std 和 corrcoef 的二维版本。

**6. 区域属性度量**

可以使用 regionprops 函数计算图像的区域属性(如面积、质心、区域边框等)。例如，对于图 6.32(a)所示的图像，使用命令：

```
BW = imread('text.tif');
L = bwlabel(BW);
stats = regionprops(L,'all');
stats(23)
```

可以得到以下的图像属性统计结果：

```
ans = Area: 89
      Centroid: [95.6742 192.9775]
      BoundingBox: [88.5000 184.5000 16 15]
      MajorAxisLength: 19.9127
      MinorAxisLength: 14.2953
      ...
      Image: [15×16 uint8 ]
      FilledImage: [15×16 uint8 ]
      FilledArea: 122
      EulerNumber: 0
      Extrema: [ 8×2 double]
      EquivDiameter: 10.6451
      Solidity: 0.4341
      ...
```

### 8.1.3 直方图灰度变换

根据以上介绍的图像统计信息可以推断出许多种图像灰度变换方法，其中最常用的就是直方图变换方法，下面将对直方图变换方法做出详细说明。

直方图是图像分析中用来说明图像灰度分布的图形，直方图的每一个分支表示对应灰度级出现的频数(即该灰度级像素个数与像素总数的比值)。根据直方图的信息可以选择一种合理的变换算法对图像进行增强操作。例如，如果一幅图像的直方图说明该图像的灰度值都较小，那么可以使用一个线性灰度调节函数来增大图像每一个像素的灰度值。

如果用变量 $f$ 代表输入图像中的像素灰度，用 $g$ 代表输出图像中的像素灰度，那么 $f$ 和 $g$ 都是数值在[0,255]范围内的连续随机变量。设输入和输出图像的灰度概率密度函数分别为 $P_f(f)$ 和 $P_g(g)$，那么所谓的直方图变换就是利用一个转移函数 $\varphi$ 对输入图像的像素灰度 $f$ 进行计算，求出输出像素灰度 $g=\varphi(f)$，使得 $P_g(g)$ 服从某种指定的概率密度分布形式，常见的有均匀分布、指数分布、双曲线分布等。根据所需的概率密度分布形式不同，采用的转移函数形式也不同。表 8.1 给出了几种不同输出图像灰度概率密度分布情况所对应的转移函数表达式，其中，输入图像灰度级个数为 $L$，$f$ 的灰度区间为 $[f_{min}, f_{max}]$，$g$ 的灰度区间为 $[g_{min}, g_{max}]$，$C(f)$ 表示输入图像灰度的累计分布函数，其定义如下：

$$C(f)=\sum_{j=0}^{k}P_f(f_j)\qquad j=0,1,\cdots,k,\cdots,L-1 \tag{8.1}$$

**表 8.1　直方图变换的转移函数形式**

| 名称 | 概率分布密度的数学模型 | 转移函数 |
| --- | --- | --- |
| 指数分布 | $P_g(g)=\alpha\exp[-\alpha(g-g_{min})]\qquad g\geq g_{min}$ | $g=g_{min}-\frac{1}{\alpha}\ln[1-C(f)]$ |
| 均匀分布 | $P_g(g)=\frac{1}{g_{max}-g_{min}}\qquad g_{max}\geq g\geq g_{min}$ | $g=[g_{max}-g_{min}]C(f)+g_{min}$ |
| Rayleigh (瑞利) | $P_g(g)=\frac{g-g_{min}}{\alpha^2}\exp\left[-\frac{(g-g_{min})^2}{2\alpha^2}\right]\qquad g\geq g_{min}$ | $g=g_{min}+\left[2\alpha^2\ln\frac{1}{1-C(f)}\right]^{1/2}$ |
| 双曲线分布 | $P_g(g)=\frac{1}{3}\left[\frac{g^{-2/3}}{g_{max}^{1/3}-g_{min}^{1/3}}\right]$ | $g=\{[g_{max}^{1/3}-g_{min}^{1/3}]C(f)+g_{min}^{1/3}\}^3$ |
| 对数双曲线分布 | $P_g(g)=\frac{1}{g[\ln(g_{max})-\ln(g_{min})]}$ | $g=g_{min}\left[\frac{g_{max}}{g_{min}}\right]^{C(f)}$ |

图像处理中经常用到的直方图均衡化就是使输出像素灰度的概率密度均匀分布的灰度变换方法。转移函数编程计算通常都是比较复杂的，在实际应用中，一般采用较为简单的转移函数形式来实现特殊要求的图像增强效果。

MATLAB 的图像处理工具箱提供一个灰度变换函数 imadjust 函数来实现图像的直方图调节。imadjust 函数的一般调用格式如下：

```
J = imadjust(I,[low_in high_in],[low_out high_out])
```

其中，low_in 和 high_in 参数分别用来指定输入图像需要映射的灰度范围，low_out 和 high_out 指定输出图像所在的灰度范围。另外，imadjust 函数还可以接受一个可选的参数来指定修正因数 $\gamma$，根据 $\gamma$ 值的不同，输入图像与输出图像间的映射可能是非线性的。图 8.6 说明了 imadjust 函数转移函数的形式和参数，缺省情况下 $\gamma=1$，表示在 low 和 high 之间的数值将会线性地映射为 bottom 和 top 之间的数值。$\gamma$ 可以是任意从 0 到无穷的数值。如果 $\gamma$ 数值为 1，那么映射将是线性的；如果 $\gamma$ 小于 1，那么映射将会对图像的像素灰度值加权，使输出像素灰度值比原来大；如果 $\gamma$ 大于 1，那么映射加权后的灰度值比原来小。图 8.6 给出了 $\gamma$ 小于 1、等于 1 和大于 1 时的数值映射曲线，其中 x 轴代表输入图像的灰度值，y 轴代表输出图像的灰度值。

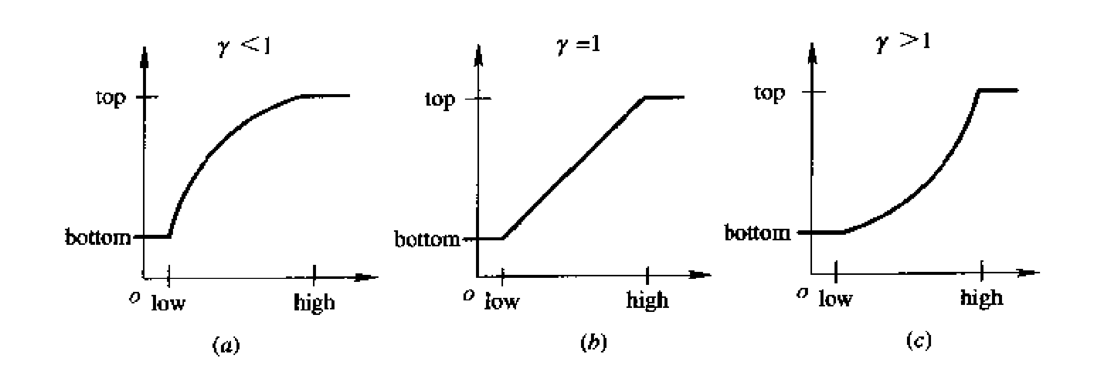



图 8.6 不同 $\gamma$ 值对应的转移函数曲线
(*a*) $\gamma<1$ 时；(*b*) $\gamma=1$ 时；(*c*) $\gamma>1$ 时

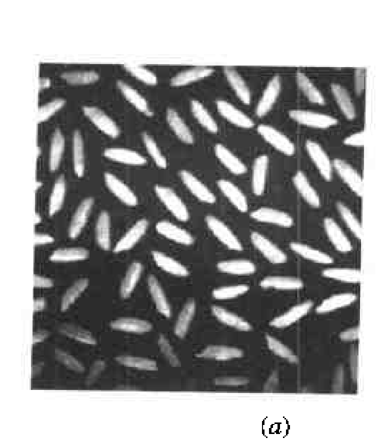
下面通过一个例子来说明 imadjust 函数的使用方法。从图 8.5(*a*)中可以看出，图像 tice.tif 是一个对比度较低的图像，可以使用直方图灰度变换来改善图像的对比度，该图像的直方图参见图 8.5(*b*)。从图 8.5(*b*)中可以看出，该图像的灰度值全部位于 40～255 之间。下面使用 imadjust 函数将图像的灰度值重新进行映射，使之填满整个灰度值允许的范围[0,255]，其代码为：

```
I = imread('rice.tif');
J = imadjust(I,[0.15 0.9],[0 1]);
imshow(J)
figure, imhist(J,64)
```

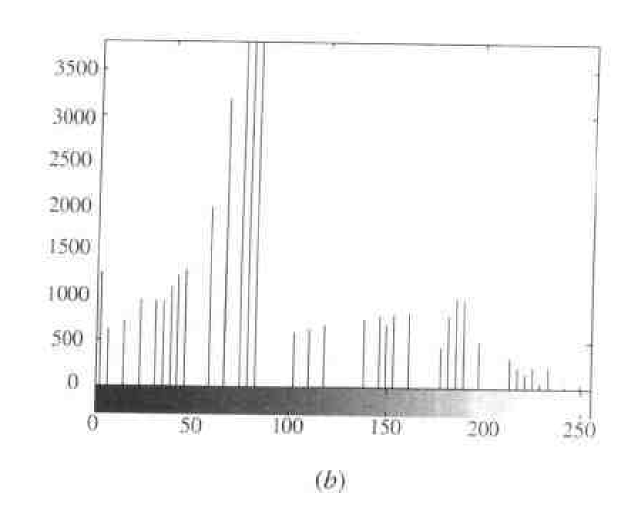
其中，imadjust 函数的第二个向量[0.15 0.9]指定需要映射的灰度值范围，第三个向量[0 1]指定希望映射到的灰度值范围。因此，输入图像中的灰度值 0.15 将被映射为输出图像中的 0，0.9 将被映射为 1。直方图变换后的图像及其直方图如图 8.7(*a*)、(*b*)所示，从图中可以看出，变换后的直方图将填满整个灰度范围。

**注意：**

无论 I 是哪一种数据类型的，指定的灰度值必须是 0.00～1.00 范围内的数值。如果 I 是 uint8 类型的，那么真正用于判断的灰度值将是指定值乘以 255 的结果；如果 I 是 uint16 类型的，那么真正用于判断的灰度值将是指定值乘以 65 535 的结果。



(a)




(b)

图 8.7　均衡化后的图像及其直方图
(a) 变换后的图像；(b) 变换后的直方图

事实上，除了增强或减弱图像的对比度，还可以使用 imadjust 函数实现很多种类的图像增强。例如在图 8.8(a)中，人物外衣的颜色比较灰暗，不能够体现其细节。调用 imadjust 函数将图 8.8(a)所示的输入图像(uint8 类型)的灰度范围从[0,51]映射到[128,255]，则输出图像如图 8.8(b)所示，其代码为

```
I = imread('cameraman.tif');
J = imadjust(I,[0 0.2],[0.5 1]);
imshow(I)
figure, imshow(J)
```



(a)



(b)

图 8.8　亮度调节前、后图像显示效果比较
(a) 调节前；(b) 调节后

上述操作将会大大提高图像的亮度，也使得原始图像灰暗部分的动态变化范围大大增加，从而使外衣的细节更容易观察。然而，要注意由于原始图像中所有大于 51 的数值都将被映射为调节后图像中的 255，所以变换后的图像看起来像是被"洗白"了一样。

从以上的例子可以看出，使用 imadjust 函数必须按照以下两个步骤来进行：

(1) 观察图像的直方图，判断灰度范围。

（2）将灰度范围转换为 0.0～1.0 之间的分数，使得灰度范围可以通过向量[low_in high_in]传递给 imadjust 函数。

更方便地指定灰度范围的方法是使用 stretchlim 函数，该函数能够计算图像的直方图，并自动判断调节范围。stretchlim 函数以分数向量形式返回灰度范围，然后将这个向量直接传递给 imadjust 函数，其调用格式如下：

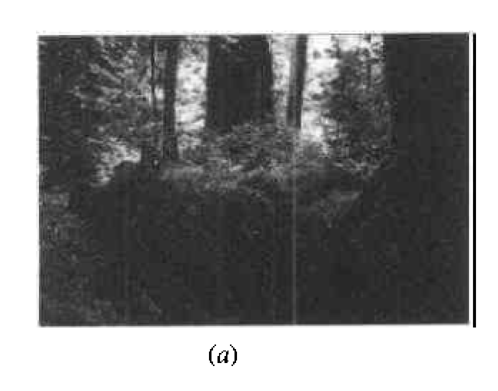
```
low=stretchlim(I,TOL)
```

其中，I 为输入图像，TOL 是一个可选的两元素向量，用来指定需要映射的灰度，缺省情况下为[0.01 0.99]。例如：

```
I = imread('rice.tif');
J = imadjust(I,stretchlim(I),[0 1]);
```

下面来看 imadjust 函数的可选参数 $\gamma$ 的修正效果。在 MATLAB 中，如果调用函数时以空矩阵作为某一个参数，那么该参数就会使用缺省值。下例中的输入、输出灰度范围都是空矩阵，那么这两个范围就会使用缺省值[0,1]，这就意味着 imadjust 函数不会改变图像的灰度范围，从而可以更好地观察 $\gamma$ 修正的效果。取 $\gamma=0.5$，图像变换前、后的显示效果如图 8.9 所示。

```
[X,map] = imread('forest.tif')
I = ind2gray(X,map);
J = imadjust(I,[],[],0.5);
imshow(I)
figure, imshow(J)
```


(*a*)

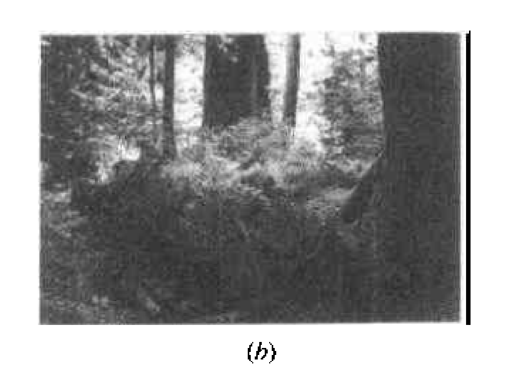

(*b*)

图 8.9 $\gamma$ 修正前、后的图像显示效果对比
(*a*) 修正前；(*b*) 修正后

### 8.1.4 直方图均衡化

直方图均衡化是一种使输出图像直方图近似为均匀分布的变换算法，其计算步骤如下：

（1）列出原始图像的灰度级 $f_j$，$j=0, 1, \cdots, k, \cdots, L-1$，其中 $L$ 是灰度级的个数。

（2）统计各灰度级的像素数目 $n_j$，$j=0, 1, \cdots, k, \cdots, L-1$。

（3）计算原始图像直方图各灰度级的频数 $P_f(f_j)=\frac{n_j}{n}$，$j=0, 1, \cdots, k, \cdots, L-1$，其中 $n$ 为原始图像总的像素数目。

(4) 计算累计分布函数 $C(f)=\sum_{j=0}^{k}P_f(f_j)$, $j=0, 1, \cdots, k, \cdots, L-1$。

(5) 应用以下公式计算映射后的输出图像的灰度级 $g_i$, $i=0, 1, \cdots, k, \cdots, P-1$, $P$ 为输出图像灰度级的个数：

$$g_i = \mathrm{INT}[(g_{max} - g_{min})C(f) + g_{min} + 0.5] \tag{8.2}$$

其中，INT 为取整符号。

(6) 统计映射后各灰度级的像素数目 $n_i$, $i=0, 1, \cdots, k, \cdots, P-1$ 。

(7) 计算输出图像直方图 $P_g(g_i)=\frac{n_i}{n}$, $i=0, 1, \cdots, k, \cdots, P-1$。

(8) 用 $f_j$ 和 $g_i$ 的映射关系修改原始图像的灰度级，从而获得直方图近似为均匀分布的输出图像。

表 8.2 给出了一幅图 8.10(*a*)所示图像的直方图均衡化的计算过程(取 $P=8$)，得到的输出图像如图 8.10(*b*)所示。

**表 8.2　图 8.10(*a*)所示图像的直方图均衡化计算过程列表**

| 已知 $f_j$ | 统计 $n_j$ | 计算 $P_f$ | 计算 $C$ | 求 $g_i$ | 统计 $n_i$ | 计算 $P_g$ | 修改原始图像 |
|---|---|---|---|---|---|---|---|
| 0 | 8 | 0.125 | 0.125 | 1 | 8 | 0.125 | 0→1 |
| 1 | 0 | 0.0 | 0.125 | 1 | 0 | 0.0 | |
| 2 | 0 | 0.0 | 0.125 | 1 | 0 | 0.0 | |
| 3 | 0 | 0.0 | 0.125 | 1 | 0 | 0.0 | |
| 4 | 31 | 0.484 | 0.609 | 4 | 31 | 0.484 | 4→4 |
| 5 | 16 | 0.25 | 0.859 | 6 | 16 | 0.25 | 5→6 |
| 6 | 8 | 0.125 | 0.984 | 7 | 9 | 0.141 | 8.7→7 |
| 7 | 1 | 0.016 | 1.0 | 7 | 9 | 0.141 | |

```
4  4  4  4  4  4  4  0        4  4  4  4  4  4  4  1
4  5  5  5  5  5  4  0        4  6  6  6  6  6  4  1
4  5  6  6  6  5  4  0        4  6  7  7  7  6  4  1
4  5  6  7  6  5  4  0        4  6  7  7  7  6  4  1
4  5  6  6  6  5  4  0        4  6  7  7  7  6  4  1
4  5  5  5  5  5  4  0        4  6  6  6  6  6  4  1
4  4  4  4  4  4  4  0        4  4  4  4  4  4  4  1
4  4  4  4  4  4  4  0        4  4  4  4  4  4  4  1
```

图 8.10　直方图均衡化前、后图像数据的比较

(a) 输入图像数据；(b) 均衡化后的输出图像数据

在 MATLAB 中，可以调用函数 histeq 自动完成图像的直方图均衡化。对于灰度图像，histeq 函数的调用格式如下：

```
J = histeq(I,n)
```

其中，n 表示输出图像的灰度级数目，是一个可选参数，缺省值为 64。

对于索引图像，其调用格式如下：

```
newmap = histeq(X,map)
```

索引图像的返回值 newmap 将是输出图像的调色板。

以下代码说明了如何使用 histeq 函数来调节一幅灰度图像：

```
I = imread('pout.tif');
J = histeq(I);
subplot(1,2,1),imshow(I)
subplot(1,2,2), imshow(J)
```

图 8.11(*a*)和图 8.11(*b*)的直方图分别如图 8.12(*a*)和图 8.12(*b*)所示。

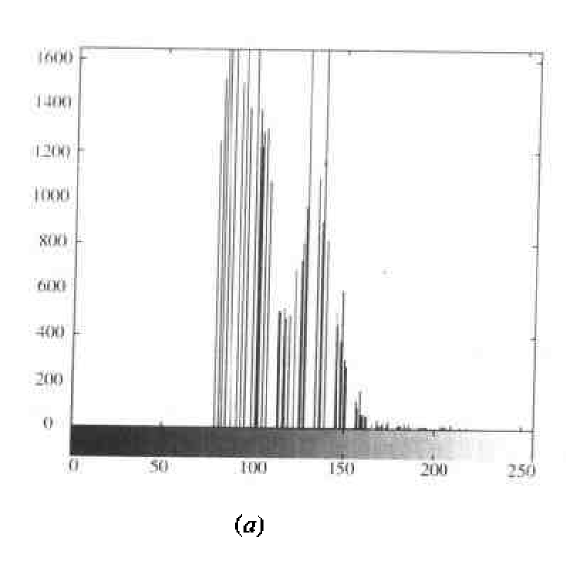


(*a*)

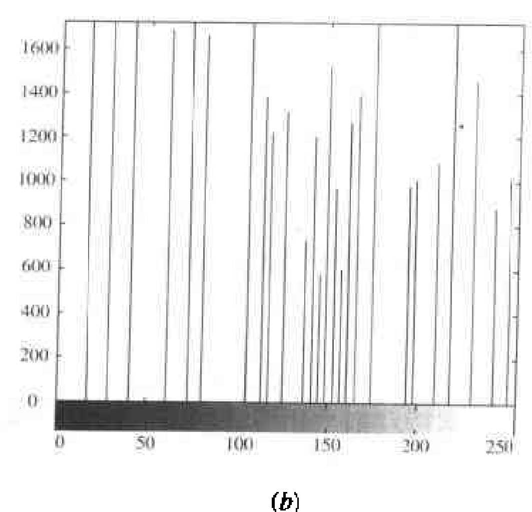


(*b*)

图 8.11 直方图均衡化前、后图像显示效果比较
(*a*) 均衡化前；(*b*) 均衡化后




(*a*)




(*b*)

图 8.12 均衡化前、后的直方图比较
(*a*) 均衡化前；(*b*) 均衡化后

可以根据 histeq 函数的返回值绘制转移函数的变换曲线，例如，以下代码将绘制如图 8.13 所示的转移函数变换曲线：

```
I = imread('pout.tif');
[J,T] = histeq(I);
figure,plot((0:255)/255,T);
```

在某些情况下，为了增强图像中某些灰度级的范围，输出图像的直方图是人为设计的，并且很难用数学模型来描述，此时可以使用直方图规定化方法来增强图像。histeq 函数可通过对灰度值进行变换实现直方图规定化，使输出图像的直方图自动匹配指定的直方

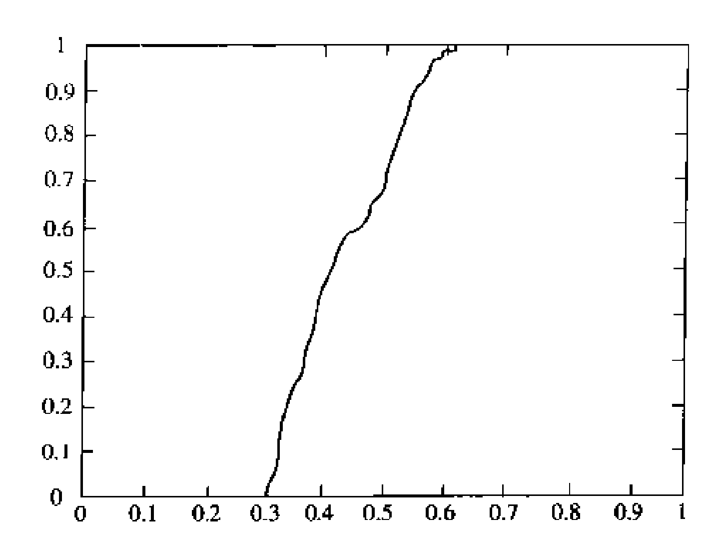



图 8.13　直方图均衡化转移函数曲线

图。此时，对于灰度图像，函数的调用格式如下：

```
J = histeq(I,n,hgram)
```

对于索引图像，其调用格式如下：

```
newmap = histeq(X,map,hgram)
```

其中，hgram 为指定的直方图向量，其长度代表直方图的柱数，每一个数值代表每一柱的像素数目，n 表示离散等级。

# 8.2　空域滤波增强

## 8.2.1　空域滤波原理及分类

空域滤波是在图像空间中借助模板对图像进行邻域操作的，输出图像每一个像素的取值都是根据模板对输入像素相应邻域内的像素值进行计算得到的。空域滤波器有很多种，它们的基本特点都是让图像在傅立叶空间某个范围内的分量受到抑制，同时保持其他分量不变，从而改变输出图像的频率分布，达到增强图像的目的。

虽然各种滤波器的类型有所不同，但是在空域中都是利用模板卷积实现这些功能的。卷积实际上就是一种使每一个输出像素取值为输入像素邻域像素值加权和的邻域操作，具体的权值是通过卷积核(也称为滤波器)定义的。例如，假设图像矩阵(参见图 8.14)为

```
A = [ 17 24 1 8 15
      23 5 7 14 16
      4 6 13 20 22
      10 12 19 21 3
      11 18 25 2 9]
```

| 17 | 24 | $1^2$ | $8^9$ | $15^4$ |
|---|---|---|---|---|
| 23 | 5 | $7^7$ | $14^5$ | $16^3$ |
| 4 | 6 | $13^6$ | $20^1$ | $22^8$ |
| 10 | 12 | 19 | 21 | 3 |
| 11 | 18 | 25 | 2 | 9 |

图 8.14　滤波卷积操作实现示意图

卷积核(图 8.14 阴影部分中像素的上标)为

h =[8 1 6
3 5 7
4 9 2]

那么，可以按照以下步骤计算输出像素(2,4)的取值：

(1) 按照卷积核的中心元素将其旋转180°。

(2) 将卷积核的中心位置移动到矩阵A的元素(2,4)处。

(3) 将旋转后卷积核的每一个权都乘以下面矩阵A的像素值。

(4) 计算步骤(3)所得的单个乘积之和。

通过以上计算得出输出像素(2,4)的取值为

$$1\times2+8\times9+15\times4+7\times7+14\times5+16\times3+13\times6+20\times1+22\times8=575$$

另外，还有一种滤波实现方法称为相关性计算。相关性操作与卷积操作密切相关。在相关性操作中，输出像素的取值也是通过计算像素邻域的加权和得到的，不同之处就在于权值矩阵。在相关性操作中的权值矩阵称为相关性核，在计算过程中不进行旋转。假设相关性核与卷积核相同，按照相关性操作后输出像素(2.4)的取值如下：

$$1\times8+8\times1+15\times6+7\times3+14\times5+16\times7+13\times4+20\times9+22\times2=585$$

根据模板的特点可以将空域滤波分为线性和非线性两类。线性空域滤波常常是基于傅立叶分析的，而非线性空域滤波通常是直接对邻域进行操作的。按照空域滤波器的功能又可以将空域滤波器分为平滑滤波器和锐化滤波器两种。平滑滤波器可以用低通滤波实现，目的在于模糊图像(提取图像中的较大对象而消除小对象或将对象的小间断连接起来)或消除图像噪声；锐化滤波器是用高通滤波实现的，目的在于强调图像被模糊的细节。下面分别介绍平滑滤波器和锐化滤波器的使用方法和用途。

## 8.2.2 平滑滤波器

### 1. 线性平滑滤波器

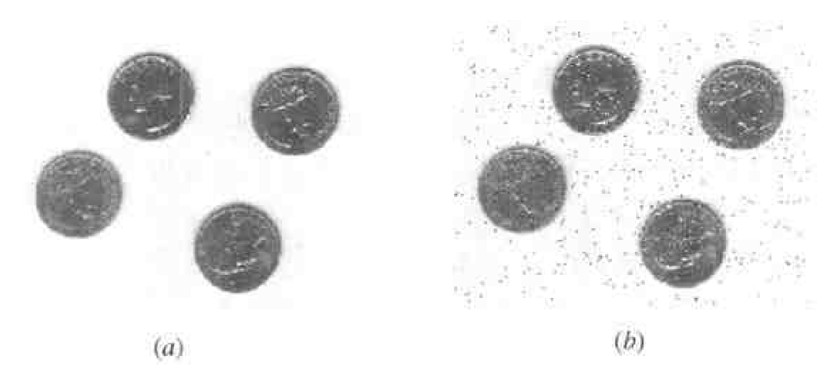
线性平滑滤波器也称为均值滤波器，是一种最常用的线性低通滤波器。均值滤波器所有的系数都是正数，为了保持输出图像仍在原来的灰度值范围内，模板与像素邻域的乘积和要除以9。以3×3邻域为例，假设当前的待处理像素为$f(m,n)$，最简单的一种均值滤波器模板如下：

$$H=\frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}、$$

将以上的均值滤波器加以修正，可以得到加权平均滤波器。例如：

$$H_1=\frac{1}{10}\begin{bmatrix}1&1&1\\1&2&1\\1&1&1\end{bmatrix}\quad H_2=\frac{1}{16}\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix}\quad H_3=\frac{1}{8}\begin{bmatrix}1&1&1\\1&0&1\\1&1&1\end{bmatrix}\quad H_4=\frac{1}{2}\begin{bmatrix}0&\frac{1}{4}&0\\\frac{1}{4}&1&\frac{1}{4}\\0&\frac{1}{4}&0\end{bmatrix}$$

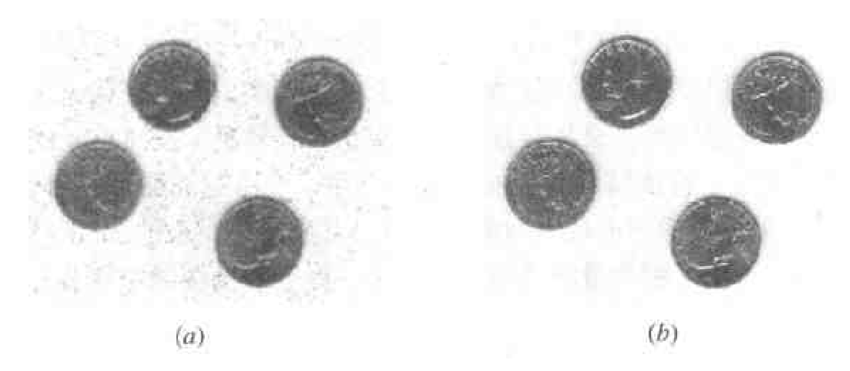
图 8.15(*a*)和图 8.15(*b*)分别给出了一幅图像及其添加了椒盐噪声后的图像，对有噪声的图像使用均值滤波器的效果如图 8.16(*a*)所示。



(*a*)　　(*b*)

图 8.15　原始图像与添加椒盐噪声图像比较

(*a*) 原图像；(*b*) 椒盐噪声图像



(*a*)　　(*b*)

图 8.16　均值滤波与中值滤波效果比较

(*a*) 均值滤波后；(*b*) 中值滤波后

还有一种常用的线性平滑滤波器是自适应滤波器，这种滤波器能够计算每一个像素的邻域统计信息，然后根据这些信息采用基于像素的自适应魏纳滤波方法对图像进行滤波处理，这种处理方法对图像固有频率附加噪声的处理效果非常好。

**2. 非线性平滑滤波器**

中值滤波器是一种最常用的非线性平滑滤波器，其滤波原理与均值滤波器方法类似，二者的不同之处在于：中值滤波器的输出像素值是由邻域像素的中间值而不是平均值决定的。中值对极限像素值(与周围像素灰度值相差较大的像素)远不如平均值那么敏感，所以中值滤波器产生的模糊较少，更适合于消除图像的孤立噪声点。

中值滤波器实际上是百分比滤波器的一种。百分比滤波器首先将模板对应的像素灰度值进行排序，然后按照确定的百分比选取灰度序列中相应的像素来作为模块中心位置对应像素的灰度值。如果百分比取 50%，那么百分比滤波器就是中值滤波器；如果百分比取最大，那么百分比滤波器就是最大值滤波器；如果取最小，那么就是最小值滤波器。

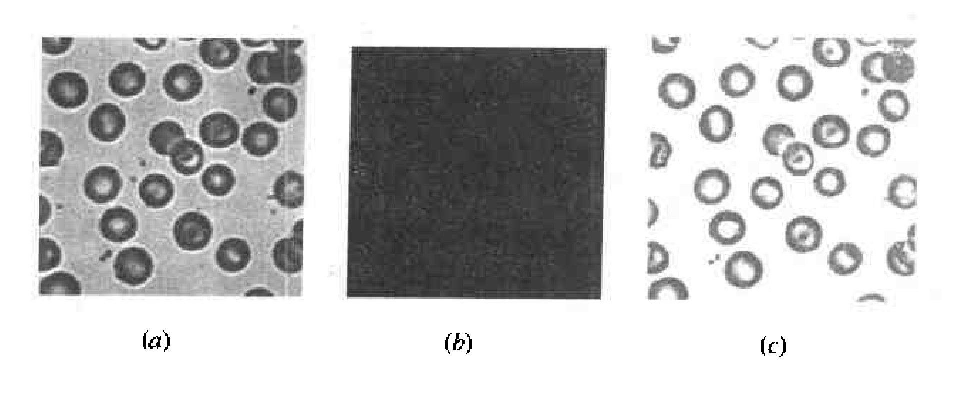
图 8.16(*b*)是对图 8.15(*b*)进行中值滤波的结果，比较图 8.16(*a*)和 8.16(*b*)可以看出，当噪声为孤立点形式时，中值滤波的效果确实要比均值滤波好。

## 8.2.3 锐化滤波器

### 1. 线性锐化滤波器

线性高通滤波器是最常用的线性锐化滤波器，这种滤波器的中心系数为正数，其他系数都为负数。这种滤波器有时会导致输出像素的灰度值为负数，而图像处理中一般仅考虑正灰度值，所以在这种情况下还要再进行灰度变换，使像素的灰度值保持在正数范围内。

高通滤波器的滤波效果也可以用原始图像减去低通图像得到。另外，如果给原始图像乘以一个放大系数，然后再减去低通图像就可以构成一幅高频增强图像，这样的图像恢复了部分高通滤波时丢失的低频成分，使得最终的结果与原始图像更为相近，这种操作也称为(非锐化)掩模。例如，图 8.17($b$)和图 8.17($c$)分别给出了图 8.17($a$)所示图像的线性高通滤波和高频增强效果。



($a$)　($b$)　($c$)

图 8.17　图像线性锐化滤波前、后显示效果示意图
($a$) 原图像；($b$) 线性高通滤波后；($c$) 掩模操作后

### 2. 非线性锐化滤波器

非线性锐化滤波就是使用微分对图像进行处理，以此来锐化由于邻域平均(相当于积分)导致的图像模糊。图像处理中最常用的微分是利用图像沿某方向上的灰度变化率，即原图像函数的梯度。梯度的定义如下：

$$grad[f(x,y)] = \left[\frac{\partial f}{\partial x} \quad \frac{\partial f}{\partial y}\right] \equiv \nabla f \tag{8.3}$$

梯度模的表达式如下：

$$|\nabla f| = \left[\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2\right]^{\frac{1}{2}} \tag{8.4}$$

$$|\nabla f| = [(\Delta_x f)^2 + (\Delta_y f)^2]^{\frac{1}{2}} \tag{8.5}$$

式(8.5)是式(8.4)的差分形式，其中

$$\Delta_x f = \frac{\Delta f}{\Delta x} = f(x+1,y) - f(x,y)$$

$$\Delta_y f = \frac{\Delta f}{\Delta y} = f(x,y+1) - f(x,y)$$

或

$$\Delta_x f = f(x,y) - f(x-1,y)$$

$$\Delta_y f = f(x,y) - f(x,y-1)$$

有时也称式(8.4)和式(8.5)为锐化的梯度算子。另外一种常用的锐化算子——拉普拉斯算子同样能够反映像素点及其邻域的灰度变化率，其定义形式为

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{8.6}$$

其差分形式为

$$\nabla^2 f = \Delta_x^2 f + \Delta_y^2 f \tag{8.7}$$

其中

$$\Delta_x^2 f = \Delta_x f(x+1,y) - \Delta_x f(x,y)$$

$$= f(x+1,y) + f(x-1,y) - 2f(x,y)$$

$$\Delta_y^2 f = f(x,y+1) + f(x,y-1) - 2f(x,y)$$

在实际处理中，常用式(8.5)和式(8.7)进行计算。

### 8.2.4 空域滤波的 MATLAB 实现方法

**1. 噪声模拟**

图像增强操作主要是针对图像的各种噪声而言的，为了说明以上滤波方法的用途，需要模拟数字图像的各种噪声来分析滤波效果。数字图像产生噪声的途径有很多种，具体依赖于图像的数字化方式。例如，如果图像是由电影图片扫描而成的，那么电影颗粒就是一个噪声源。噪声也可以是电影遭到破坏的结果，或者是由扫描仪自身产生的。如果图像直接以数字形式获得，那么收集数据的机制(例如，CCD 扫描仪)将会不可避免地引入噪声。另外，图像数据的传输也会引入噪声。

MATLAB 的图像处理工具箱提供 imnoise 函数，可以用该函数给图像添加不同种类的噪声，该函数的调用格式如下：

```
J = imnoise(I,'type',parameters)
```

表 8.3 列出了 imnoise 函数能够产生的五种噪声及其对应参数。

**表 8.3　imnoise 函数支持的噪声种类及参数说明**

| type | parameters | 说　明 |
|---|---|---|
| gaussian | m,v | 均值为 m，方差为 v 的高斯噪声 |
| localvar | v | 均值为 0，方差为 v 的高斯白噪声 |
| possion | 无 | 泊松噪声 |
| salt&pepper | 无 | 椒盐噪声 |
| speckle | v | 均值为 0，方差为 v 的均匀分布随机噪声 |

**2. MATLAB 预定义滤波器**

可以调用 fspecial 函数来创建 MATLAB 预定义的滤波器，其调用格式如下：

```
h = fspecial('type',parameters)
```

其中，参数 type 指定滤波器的种类，parameters 是与滤波器种类有关的具体参数。表 8.4 列出了 MATLAB 预定义滤波器的种类及其对应参数。

**表 8.4　MATLAB 预定义滤波器的种类及 fspecial 函数参数**

| type | parameters | 说　明 |
|---|---|---|
| gaussian | hsize,sigma | 标准偏差为 sigma、大小为 hsize 的高斯低通滤波器 |
| sobel | 无 | 使用近似计算垂直梯度光滑效应的水平边缘强调算子 |
| prewitt | 无 | 近似计算垂直梯度的水平边缘强调算子 |
| laplacian | alpha | 系数由 alpha(0.0～1.0)决定的二维拉普拉斯操作 |
| log | hsize,sigma | 标准偏差为 sigma、大小为 hsize 的高斯滤波旋转对称拉氏算子 |
| disk | radius | 有(radius＊2＋1)个边的圆形均值滤波器 |
| average | hsize | 均值滤波器，如果邻域为方阵，则 hsize 为标量，否则由两元素向量 hsize 指定邻域的行数和列数 |
| unsharp | alpha | 根据 alpha 决定的拉氏算子创建的掩模滤波器 |
| motion | len,theta | 按照角度 theta 移动 len 个像素的运动滤波器 |

**3. 图像滤波**

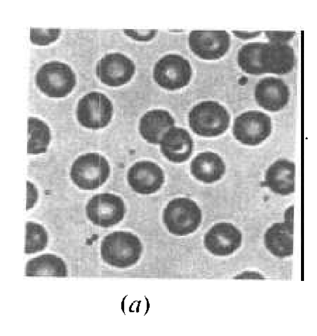
可以使用 imfilter 函数或 filter2 函数调用创建好的滤波器(可以是预定义滤波器，也可以是自定义滤波器)对图像进行滤波。本书将重点介绍图像处理工具箱函数 imfilter 函数的使用方法。

卷积或相关性滤波都可以使用工具箱函数 imfilter 实现。例如，对图像 blood1.tif(如图 8.18(*a*)所示)使用一个权值全部为 1 的 5×5 滤波器进行均值滤波：

```
I = imread('blood1.tif');
h = ones(5,5)/ 25;
I2 = imfilter(I,h);
subplot(1,2,1),imshow(I)
subplot(1,2,2), imshow(I2)
```

滤波结果如图 8.18(*b*)所示。

imfilter 函数使用与图像代数运算函数相同的方法控制数据类型，输出图像与输入图像有相同的数据类型和格式。imfilter 函数使用双精度浮点算术计算每一个输出像素的数值，如果结果超过数据类型的范围，那么 imfilter 函数将按照数据类型允许的数据范围对结果进行截取；如果数据类型是整数，那么 imfilter 将会舍去分数部分。考虑到截取的效果，有时在调用 imfilter 函数之前可能需要将图像转换为另一种数据类型。例如，当输入数据是双精度类型时，调用 imfilter 函数产生的输出将会出现负值。


(a)

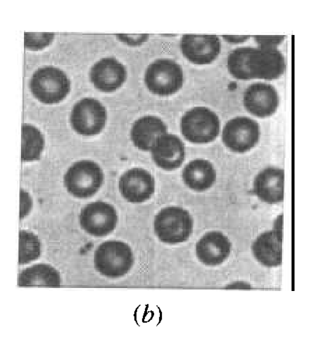

(b)

图 8.18　使用 imfilter 函数进行均值滤波前、后的显示效果对比
(a) 原图像；(b) 均值滤波后

```
A = magic(5);
h = [-1 0 1];
imfilter(A,h)
ans =
     24   -16   -16    14    -8
      5   -16     9     9   -14
      6     9    14     9   -20
     12     9     9   -16   -21
     18    14   -16   -16    -2
```

假设矩阵 A 是 uint8 类型的，那么由于 imfilter 函数的输入是 uint8 类型的，所以输出也是 uint8 类型的，而且所有负值都被截取为 0。

```
A = uint8(magic(5));
imfilter(A,h)
ans =
     24    0    0   14    0
      5    0    9    9    0
      6    9   14    9    0
     12    9    9    0    0
     18   14    0    0    0
```

可见，在这种情况下，在调用 imfilter 函数之前需将图像转换为其他类型，例如，转换为有符号整数类型或双精度类型可能更为合适。

imfilter 函数既能够使用卷积，也能够使用相关性来进行滤波，一般在缺省情况下使用相关性方法，因为滤波器设计函数以及 fspecial 函数生成的都是相关性核。如果用户希望使用卷积方法实现滤波，将字符串 conv 作为输入参数传递给 imfilter 函数，则语句表示如下：

```
imfilter(A,h,'conv')      % 使用卷积滤波
ans =
    -24    16    16   -14     8
     -5    16    -9    -9    14
     -6    -9   -14    -9    20
    -12    -9    -9    16    21
    -18   -14    16    16     2
```

这里要注意这样一个问题：当计算一幅图像边界的输出像素时，卷积核以及相关性核的一部分通常都会超出图像边界，如图 8.19(*a*)所示。imfilter 函数通常假设这些不存在的像素为 0，这种方法称为零填充，填充结果如图 8.19(*b*)所示。

| | | $?^8$ | $?^1$ | $?^6$ |
|---|---|---|---|---|
| 17 | 24 | $1^3$ | $8^5$ | $15^7$ |
| 23 | 5 | $7^4$ | $14^9$ | $16^2$ |
| 4 | 6 | 13 | 20 | 22 |
| 10 | 12 | 19 | 21 | 3 |
| 11 | 18 | 25 | 2 | 9 |

(*a*)

| | | $0^8$ | $0^1$ | $0^6$ |
|---|---|---|---|---|
| 17 | 24 | $1^3$ | $8^5$ | $15^7$ |
| 23 | 5 | $7^4$ | $14^9$ | $16^2$ |
| 4 | 6 | 13 | 20 | 22 |
| 10 | 12 | 19 | 21 | 3 |
| 11 | 18 | 25 | 2 | 9 |

(*b*)

| | | $1^8$ | $8^1$ | $15^6$ |
|---|---|---|---|---|
| 17 | 24 | $1^3$ | $8^5$ | $15^7$ |
| 23 | 5 | $7^4$ | $14^9$ | $16^2$ |
| 4 | 6 | 13 | 20 | 22 |
| 10 | 12 | 19 | 21 | 3 |
| 11 | 18 | 25 | 2 | 9 |

(*c*)

图 8.19　边界填充示意图

(*a*) 相关性核超出图像边界；(*b*) 零填充效果；(*c*) 边界复制效果

对一幅图形进行滤波时，零填充可能会导致图像被一个黑框围绕，例如图 8.18(*b*)所示。为了消除零填充的人工痕迹，imfilter 函数还支持三种可选的边界填充方法：边界复制(replicate)、边界循环(circular)、边界对称(symmetric)。以边界复制为例，所有位于图像外部像素的取值都是通过复制最近的边界像素值获得的。图 8.19(*c*)说明了这个过程。使用边界复制方式进行滤波时，将参数 replicate 传递给 imfilter 函数：

```
I3 = imfilter(I,h,'replicate');
figure, imshow(I3), title('Filtered with border replication')
```

此时对图 8.18(*a*)进行滤波得到的结果如图 8.20 所示。

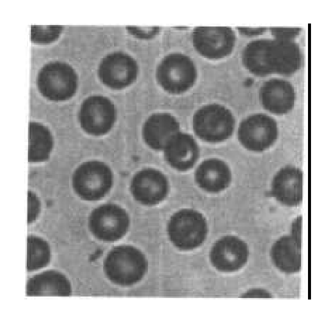


图 8.20　边界复制填充方法的滤波效果

除了 imfilter 函数以外，MATLAB 还有许多二维和多维滤波函数，例如，filter2、conv2、convn、medfilt2 等，这些函数总是要将输入转换为双精度类型，输出也总是双精度类型的，并且仅仅支持零填充方法。例如，图 8.16 所示的两幅滤波图像可以使用以下步骤实现：

首先，将图像读取出来，并添加噪声。

```
I = imread('eight.tif');
J = imnoise(I,'salt & pepper',0.02);
```

然后，分别对图像进行均值滤波和中值滤波并显示结果。

```
K = filter2(fspecial('average',3),J)/255;
L = medfilt2(J,[3 3]);
subplot(1,2,1), imshow(K)
subplot(1,2,2), imshow(L)
```

前面已经介绍过，MATLAB 的 wiener2 函数可以对图像进行自适应的魏纳滤波，这种

滤波方法能够改变自身模板的大小以符合局部图像变量。当变量值很大时，wiener2 将执行小平滑过程；当变量值较小时，wiener2 将执行大面积的平滑操作，这种方法通常能够产生比线性滤波更好的效果。wiener2 函数需要的计算时间比线性滤波器的要长。wiener2 函数在噪声为固定功率噪声(即白噪声，例如高斯噪声)时，其滤波效果最好。

## 8.3 频 域 增 强

### 8.3.1 低通滤波

一般来说，图像的边缘和噪声都对应于傅立叶变换中的高频部分，所以能够让低频信息畅通无阻而同时滤掉高频分量的低通滤波器能够平滑图像，去除噪声。常用的几种频域低通滤波器有理想低通滤波器、巴特沃斯(Butterworth)低通滤波器、指数低通滤波器，其传递函数的形式分别如式(8.8)、(8.9)和(8.10)所示。

理想低通滤波器：

$$H(u,v)=\begin{cases}1 & D(u,v)\leqslant D_0\\ 0 & D(u,v)>D_0\end{cases} \tag{8.8}$$

巴特沃斯低通滤波器：

$$H(u,v)=\frac{1}{1+(\sqrt{2}-1)\left[\frac{D(u,v)}{D_0}\right]^{2n}} \tag{8.9}$$

指数低通滤波器：

$$H(u,v)=e^{-\frac{[D(u,v)/D_0]^n}{\sqrt{2}}} \tag{8.10}$$

其中，$D(u,v)=\sqrt{u^2+v^2}$，表示点$(u,v)$到原点的距离，$D_0$ 表示截止频率点到原点的距离。

这里应该指出，傅立叶变换的主要能量都是集中在频谱中心的，合理地选择截止频率对保留图像的能量是至关重要的。以一幅 256×256 的图像为例，如果 $D_0=5$，那么理想低通滤波器将保存图像 90%的能量。随着 $D_0$ 的增大，图像的能量将迅速流失，如果 $D_0=22$，那么 98%的能量将会通过该滤波器流失。另外，理想低通滤波后的图像将会出现一种“振铃”特性，造成图像不同程度的模糊，$D_0$ 越小，模糊程度越明显。造成这种模糊的原因在于理想低通滤波器的传递函数 $H(u,v)$在 $D_0$ 处由 1 突变为 0，该 $H(u,v)$经过傅立叶反变换后在空域中将表现为同心圆的形式。

### 8.3.2 高通滤波

由于图像中灰度发生骤变的部分与其频谱的高频分量相对应，所以采用高通滤波器衰减或抑制低频分量，使高频分量畅通并能够对图像进行锐化处理。常用的高通滤波器有理想高通滤波器、巴特沃斯高通滤波器、指数高通滤波器，其传递函数分别如式(8.11)、(8.12)和(8.13)所示。

理想高通滤波器：

$$H(u,v)=\begin{cases}0 & D(u,v)\leqslant D_0\\ 1 & D(u,v)>D_0\end{cases} \tag{8.11}$$

巴特沃斯高通滤波器：

$$H(u,v) = \frac{1}{1 + (\sqrt{2} - 1)\left[\frac{D_0}{D(u,v)}\right]^{2n}} \tag{8.12}$$

指数高通滤波器：

$$H(u,v) = e^{-\frac{[D_0/D(u,v)]^n}{\sqrt{2}}} \tag{8.13}$$

由于经过高通滤波处理后的图像丢失了许多低频信息，所以图像的平滑区基本会消失。为此，需要采用高频加强滤波来弥补。高频加强滤波就是在设计滤波传递函数时，在原有的设计结果上添加一个大于0、小于1的常数 $c$，即

$$H'(u,v) = H(u,v) + c \tag{8.14}$$

于是滤波结果为

$$G'(u,v) = F(u,v)H'(u,v) = F(u,v)H(u,v) + cF(u,v) \tag{8.15}$$

由式(8.15)可见，高频加强滤波在高通滤波的基础上保留了 $cF(u,v)$ 的低通分量，高频分量也比一般高通滤波时加强了 $cF(u,v)$，故称为高频加强滤波，这种滤波处理效果比一般的高通滤波要好。

### 8.3.3 同态滤波

同态滤波是一种在频域中同时进行图像对比度增强和压缩图像亮度范围的特殊滤波方法。同态系统是指服从广义叠加原理的，输入和输出之间可以用线性变化表示的系统。如果输入量为乘法运算组合形式，则称该系统为乘法同态系统。

图像处理中的同态滤波是基于以反射光和入射光为基础的图像模型的。如果把图像亮度 $f(x,y)$ 看成是由入射分量（入射到景物上的光强度）$i(x,y)$ 和反射分量（景物反射的光强度）$r(x,y)$ 组成的，那么图像的模型可以表示为

$$f(x,y) = i(x,y) \cdot r(x,y) \quad 0 < i(x,y) < \infty,\ 0 < r(x,y) < 1 \tag{8.16}$$

从式(8.16)可以看出，图像适合作为乘法同态系统处理。进行图像增强处理的同态滤波过程是这样的：首先对图像取对数，使图像模型中的乘法运算组合变成简单的对数加法运算组合，然后对对数图像作傅立叶变换，再选择合适的传递函数进行滤波，最后对滤波结果进行傅立叶反变换和对数逆运算（即指数运算），得到同态滤波后的输出结果。

从同态滤波的实现过程可以看出，能否达到预期的增强效果并取得压缩灰度的动态范围的效果取决于同态滤波传递函数的选择。人们通过对入射光分量和反射光分量的研究发现，入射光分量一般反映灰度恒定分量，类似于低频信息，减弱入射光可以缩小图像的灰度范围；而反射光与物体的边界特性密切相关，类似于高频信息，增强反射光可以提高对比度。因此，同态滤波的传递函数一般在低频部分小于1，在高频部分大于1。

### 8.3.4 频域增强的 MATLAB 实现

MATLAB 的频域增强效果也是使用 imfilter 或 filter2 函数实现的，这里要注意的是，因为一般 MATLAB 系统不提供频域增强所使用的滤波器，所以用户需要自定义这些滤波器。首先使用第五章中介绍的滤波器设计方法来创建某种类型的频域增强滤波器，然后调用 imfilter 或 filter2 函数对图像进行滤波操作，来实现图像的增强。例如，使用最优波纹低

通滤波器(其频率响应曲线如图 8.21(*c*)所示)对图 8.21(*a*)所示的图像进行滤波增强，处理后的图像如图 8.21(*b*) 所示。这个增强过程将作为习题留给读者思考。

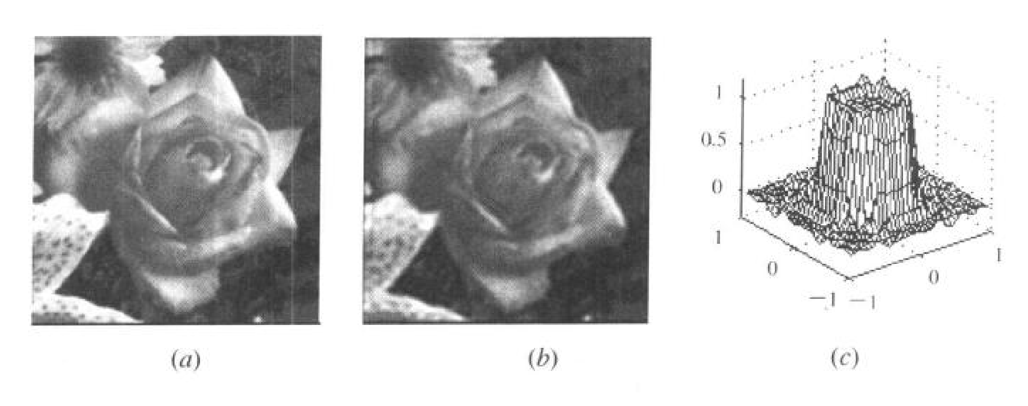



图 8.21　低通滤波器的滤波效果及其频率响应曲线
(*a*) 滤波前；(*b*) 滤波后；(*c*) 频率响应曲线

## 8.4　色 彩 增 强

### 8.4.1　色彩增强概述

在图像的自动分析中，色彩是一种能够简化目标提取和分类的重要参数。虽然人眼只能分辨几十种不同深浅的灰度级，但是却能够分辨几千种不同的颜色，因此在图像处理中常常借助色彩来处理图像，以增强人眼的视觉效果。

通常采用的色彩增强方法可以分为伪彩色增强和真彩色增强两种，这两种方法在原理上存在着巨大的差别。伪彩色增强是对原来灰度图像中不同灰度值区域分别赋予不同的颜色，使人眼能够更明白地区分不同的灰度级。由于原始图像事实上是没有颜色的，所以称这种人工赋予的颜色为伪彩色。伪彩色增强实质上只是一个图像的着色过程，是一种灰度到彩色的映射技术；真彩色增强则是对原始图像本身所具有的颜色进行调节，是一个色彩到色彩的映射过程。由此可见，二者有着本质的区别。下面将对这两种方法进行详细介绍。

### 8.4.2　伪彩色增强

伪彩色增强是一种将二维图像像素逐点映射到由三基色确定的三维色度空间中的技术，其目的在于利用人眼对色彩的敏感性，应用伪彩色技术使图像中的不同物体具有一定的色差，从而提高人对图像的分辨能力。伪彩色处理可以分为空域增强和频域增强两种。空域伪彩色处理实际上是将图像的灰度范围划分为若干等级区间，每一个区间映射为某一种颜色。空域伪彩色处理是基于频域运算基础上的伪彩色处理方法。输入图像经过傅立叶变换得到图像的频谱，然后将频谱的各个分量分别送到 R、G、B 三个通道进行滤波，最后对各通道作傅立叶反变换，得到空域的 R、G、B 分量，最终产生彩色图像。

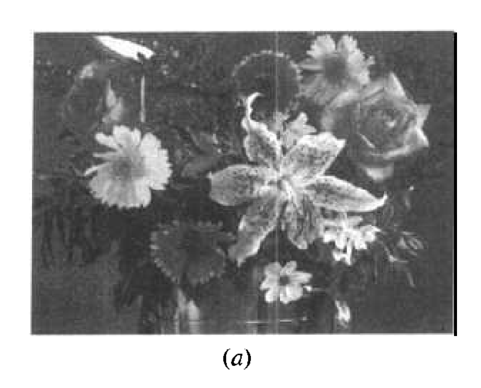
MATLAB 图像处理工具箱没有专门的图像伪彩色处理函数，但是工具箱中包含许多可以用来实现伪彩色的函数。例如，第一章中介绍的灰度图像类型转换函数 grayslice、gray2ind 等，这些函数都是使用空域增强方法来实现图像的伪彩色显示的，可以通过设置

函数的参数来选择调色板，也可以使用函数默认的调色板来进行灰度映射。

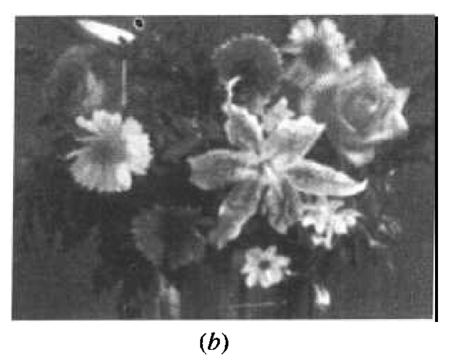
### 8.4.3 真彩色增强

在MATLAB中，调用imfilter函数对一幅真彩色(三维数据)图像使用二维滤波器进行滤波就相当于使用同一个二维滤波器对数据的每一个平面单独进行滤波。例如，以下代码将使用均值滤波器对图8.22(*a*)所示的真彩色图像的每一个颜色平面进行滤波，滤波结果如图8.22(*b*)所示。

```
rgb = imread('flowers.tif');
h = ones(5,5) / 25;
rgb2 = imfilter(rgb,h);
subplot(1,2,1),imshow(rgb)
subplot(1,2,2), imshow(rgb2)
```



(*a*)



(*b*)

图8.22 真彩色图像均值滤波前、后显示效果对比
(*a*) 滤波前；(*b*) 滤波后

## 【习 题】

1. 利用第二章中介绍的邻域操作实现图像的非线性锐化滤波。
2. 实现如图8.21所示的图像增强效果。

# 第九章 图像复原

**本章要点：**

★ 成像系统的数学描述

★ 图像退化模型

★ 图像复原的代数方法

★ 图像复原的 MATLAB 实现方法

## 9.1　成像系统的数学描述

图像是物体经过光学系统映射得到的图片。如果一个成像系统是理想的，那么映射的结果将得到一幅理想的图像，该图像能够毫不失真地映射物体上的所有信息。但是成像系统总是或多或少地存在一些缺陷，例如，光学系统的球差畸变等，摄影胶片的非线性，光电转换器件的非线性，摄像机与目标间的相对运动，遥感图像中的大气扰动等都会使图像质量下降，造成图像退化现象。

所谓图像复原就是在研究图像退化原因的基础上，以退化图像为依据，根据一定的先验知识设计一种算子，从而估计出理想场景的操作。一般得到一幅数字化图像后都会先使用图像复原技术进行处理，然后再作增强处理。由于不同应用领域的图像有不同的退化原因，所以对同一幅退化图像，不同应用领域要采用不同的复原方法。在实际应用过程中，应该本着具体问题具体分析的原则充分利用本章介绍的方法进行图像复原。

为了刻画成像系统的特征，通常将成像系统看成是一个线性系统，从中推导出物体输入与图像输出关系的通用数学表达式，从而建立成像系统的退化模型，在此基础上研究图像复原技术。事实上，成像系统总是存在一定的非线性性质的，但是如果这些非线性性质不至于引起严重的误差，或者是当成像系统在小范围内满足线性性质时，一般仍将成像系统看成是线性的，这是因为线性系统已经有了完整的理论体系，不但处理起来方便，而且能够反映系统的主要特性。

由成像系统的线性性质可知，系统中的物函数(即系统输入激励)$f(x,y)$可以分解为无数个基元物函数之和，最简单的基元物函数就是 $\delta$ 函数(脉冲函数)，因此物函数可以表示为

$$f(x,y)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}f(\xi,\eta)\delta(x-\xi,y-\eta)\mathrm{d}\xi\,\mathrm{d}\eta \tag{9.1}$$

式(9.1)表明，物函数可以看成是带有权因子的 $f(\xi,\eta)$ 的 $\delta$ 函数的线性组合。因此，只要求出成像系统对 $\delta$ 函数的响应表达式，将其与每一个基元物函数的权因子进行相乘、求和就

可以得到像平面上的像函数 $g(x,y)$。$g(x,y)$的表达式如下：

$$g(x,y) = \sigma\left\{\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\xi,\eta)\delta(x-\xi,y-\eta)\mathrm{d}\xi\,\mathrm{d}\eta\right\} \tag{9.2}$$

式(9.2)中的 $\sigma\{\cdot\}$表示成像系统的传递函数。如果用符号 $h(x,y;\xi,\eta)$表示系统对 $\delta$ 函数的响应，即

$$h(x,y;\xi,\eta) = \sigma\{\delta(x-\xi,y-\eta)\} \tag{9.3}$$

则称函数 $h(x,y;\xi,\eta)$为成像系统的脉冲响应，有时也称为点扩散函数，这是因为物体上的点经过成像系统后将不是一个点，而是一个扩散的同心圆的缘故(经夫琅和费衍射实验验证可得)。系统的输入、输出关系可以表示为

$$g(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\xi,\eta)h(x,y;\xi,\eta)\mathrm{d}\xi\,\mathrm{d}\eta \tag{9.4}$$

由于光学成像系统的脉冲响应仅依赖于$(x-\xi)$和$(y-\eta)$，即物平面的点光源在场景中移动时，点光源所成的像也只是改变位置而函数形式不变，这样就有

$$h(x,y;\xi,\eta) = h(x-\xi,y-\eta) \tag{9.5}$$

于是式(9.4)变为

$$g(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\xi,\eta)h(x-\xi,y-\eta)\mathrm{d}\xi\,\mathrm{d}\eta \equiv f * h \tag{9.6}$$

其中，符号 * 表示卷积。满足(9.5)式的点扩散函数称为空不变的点扩散函数，在几何光学中称为等晕条件，表示轴外光线造成的焦散误差应与轴上光线具有同样性质。

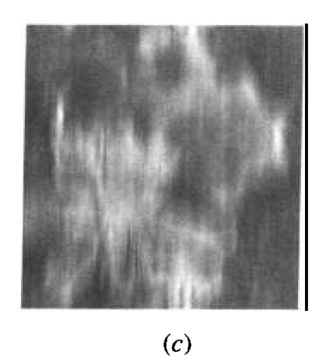
上述方程就是近代光学中用来描述成像系统的数学表达式，这些表达式说明，物函数与系统脉冲响应的卷积结果就是考虑了衍射效应后系统所成的像，衍射效应越强，退化越严重。由此可见，点扩散函数决定了系统的成像质量。这一点可以从图 9.1(*a*)、(*b*)和(*c*)所示的三幅图像中看出来。图 9.1(*a*)是一个对真实场景比较清晰的描述，而 9.1(*b*)和 9.1(*c*)则分别表示由水平运动和垂直运动点扩散函数与图像卷积形成的图像。



(*a*)



(*b*)



(*c*)

图 9.1 不同点扩散函数导致的图像模糊比较

(*a*) 原图像；(*b*) 水平点扩散后的图像；(*c*) 垂直点扩散后的图像

有了以上的成像系统数学描述形式，下一步就可以推断出图像的退化模型，然后按照某种标准尽可能地恢复场景的原始面貌。从退化图像中较为精确地找出真实图像是一个估计问题，评价估计效果的标准有很多，例如，最小二乘方准则等，不同的复原方法实际上就是针对不同估计评价标准而言的。从以上的分析也可以看出，图像复原是图像退化过程

的逆向估计，是一个去卷积的过程。

# 9.2 图像退化模型

## 9.2.1 连续退化模型

图像复原处理一定是建立在图像退化的数学模型基础上的，这个退化数学模型应该能够反映图像退化的原因。由于图像的退化因素较多，而且比较复杂，不便于逐个分析和建立数学模型，所以图像处理过程中通常是把退化原因作为线性系统退化的一个因素来对待，从而建立系统退化模型来近似描述图像函数的退化模型。

假设，$g(x,y)$代表一幅退化图像，$f(\xi,\eta)$为原图像(真实场景的描述)，退化模型可以用图 9.2 来描述。

图中 $n(x,y)$表示系统噪声。由图 9.2 建立的图像退化模型的一般表达式为

$$\begin{aligned} g(x,y) &= n(x,y) + \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\xi,\eta)h(x-\xi,y-\eta)\mathrm{d}\xi\,\mathrm{d}\eta \\ &= f(x,y) * h(x,y) + n(x,y) \end{aligned} \tag{9.7}$$

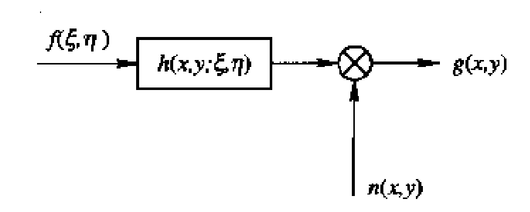



图 9.2 图像的退化模型

从式(9.7)可以看出，图像的退化就是成像系统的退化加上额外的系统噪声而形成的。根据这个模型可知，图形复原就是在退化图像的基础上已知 $h(x,y)$和 $n(x,y)$，然后进行反演运算，得到一个 $f(x,y)$的最佳估计 $\hat{f}(x,y)$。之所以说得到的是 $f(x,y)$的最佳估计而不是真实的 $f(x,y)$有两个原因：一个原因是在进行反演运算时，反演方程不一定有解，这就是图像复原中最棘手的问题——奇异问题；另一个原因是反演方程可能存在多个解。这两种情况一般称为图像复原的病态性。

## 9.2.2 离散退化模型

由于数字图像都是离散形式的，所以在实际应用中都是采用式(9.7)的离散形式进行计算的，其表达式如下：

$$g(x,y) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f(m,n)h(x-m,y-n) + n(x,y) \tag{9.8}$$

式中 $x=0, 1, 2, \cdots, M-1$；$y=0, 1, 2, \cdots, N-1$。函数 $f(x,y)$和 $h(x,y)$分别是周期为 $M$ 和 $N$ 的函数。注意，如果这两个函数的周期不是 $M$ 和 $N$，那么必须对它们进行补零延拓，避免卷积周期的交叠。$g(x,y)$是与 $f(x,y)$和 $h(x,y)$具有相同周期的函数。

以下将由 $M\times N$ 函数矩阵 $f(x,y)$、$g(x,y)$和 $h(x,y)$各行堆叠形成的 $MN$ 维列向量分别记为 $\boldsymbol{f}$、$\boldsymbol{g}$ 和 $\boldsymbol{n}$，形式如下：

$$f=\begin{bmatrix} f(0,0) \\ f(0,1) \\ \cdots \\ f(0,N-1) \\ \cdots \\ f(M-1,0) \\ f(M-1,1) \\ \cdots \\ f(M-1,N-1) \end{bmatrix} \quad g=\begin{bmatrix} g(0,0) \\ g(0,1) \\ \cdots \\ g(0,N-1) \\ \cdots \\ g(M-1,0) \\ g(M-1,1) \\ \cdots \\ g(M-1,N-1) \end{bmatrix} \quad n=\begin{bmatrix} n(0,0) \\ n(0,1) \\ \cdots \\ n(0,N-1) \\ \cdots \\ n(M-1,0) \\ n(M-1,1) \\ \cdots \\ n(M-1,N-1) \end{bmatrix}$$

则式(9.8)可以写为

$$\boldsymbol{g}=\boldsymbol{H}\boldsymbol{f}+\boldsymbol{n} \tag{9.9}$$

式中 $\boldsymbol{H}$ 为 $MN\times MN$ 维矩阵。$\boldsymbol{H}$ 包括 $M^2$ 个子矩阵，每一个子矩阵的大小为 $N\times N$，排列顺序如下：

$$\boldsymbol{H}=\begin{bmatrix} H_0 & H_{M-1} & H_{M-2} & \cdots & H_1 \\ H_1 & H_0 & H_{M-1} & \cdots & H_2 \\ H_2 & H_1 & H_0 & \cdots & H_3 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ H_{M-1} & H_{M-2} & H_{M-3} & \cdots & H_0 \end{bmatrix} \tag{9.10}$$

式(9.10)中的每一个子矩阵 $\boldsymbol{H}_j$ 都是由 $h(x,y)$ 的第 $j$ 行构成的

$$\boldsymbol{H}_j=\begin{bmatrix} h(j,0) & h(j,N-1) & h(j,N-2) & \cdots & h(j,1) \\ h(j,1) & h(j,0) & h(j,N-1) & \cdots & h(j,2) \\ h(j,2) & h(j,1) & h(j,0) & \cdots & h(j,3) \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ h(j,N-1) & h(j,N-2) & h(j,N-3) & \cdots & h(j,0) \end{bmatrix} \tag{9.11}$$

### 9.2.3 图像复原方法概述

从式(9.10)和(9.11)可以看出，$\boldsymbol{H}$ 是一个分块循环矩阵，每一个元素都是由点扩散函数表示的，因而也称为点扩散函数矩阵。从式(9.9)描述的图像退化模型来看，当给定了 $\boldsymbol{g}$，并对 $\boldsymbol{H}$ 和 $\boldsymbol{n}$ 有一定的了解后，复原问题就是要从式(9.9)中估计出 $\boldsymbol{f}$ 来，显然，这是退化过程的一个逆向计算过程，如果噪声为零，那么该过程就是一个去卷积的过程。但是从式(9.10)和(9.11)也可以看出，在实际应用中，如果 $M$ 和 $N$ 的取值较大，那么矩阵 $\boldsymbol{H}$ 将会非常庞大，使得计算工作量过大，有两个办法可以解决这个问题：一是对分块循环矩阵进行简化，再利用现有的快速算法进行求解。例如，对分块循环矩阵进行对角化，然后使用离散傅立叶变换对式(9.9)进行求解；另一种办法是分析退化的原因，找出 $\boldsymbol{H}$ 的具体函数表达形式(可以适当作些简化)。例如，在 $x$ 方向上运动产生模糊的 $\boldsymbol{H}$ 的函数表示形式为

$$h(x)=\begin{cases} T/a & 0\leqslant x\leqslant a \\ 0 & \text{其他} \end{cases} \tag{9.12}$$

式中 $a$ 表示曝光时间 $T$ 内的总位移。

# 9.3 图像复原的代数方法

## 9.3.1 基本复原方程

根据以上的讨论可知，图像复原实际上就是在已知系统点扩散函数和噪声的条件下，以最小二乘方为准则求逆变换：

$$\hat{f} = \sigma^{-1}\{g\} \tag{9.13}$$

图像的代数复原方法的基本思想是寻找一个最接近输入向量 $f$ 的估计值，这个估计值能够使得预先规定的准则取值最小。无约束代数复原方法和约束代数复原方法是两种主要的代数复原方法，下面我们分别介绍这两种方法的基本方程。

### 1. 无约束复原方程

根据式(9.9)可得：

$$n = g - Hf \tag{9.14}$$

则 $n$ 的范数的平方为

$$\|n\|^2 = n^{\mathrm{T}}n = (g - Hf)^{\mathrm{T}}(g - Hf) \tag{9.15}$$

可以将 $\|n\|$ 看成是一种噪声大小的度量。无约束复原就是求式(9.15)的最小二乘方解，即要寻找一个 $\hat{f}$，使 $\|n\|$ 最小。令

$$J(\hat{f}) = (g - H\hat{f})^{\mathrm{T}} \cdot (g - H\hat{f}) \tag{9.16}$$

则无约束复原方法就转化为求式(9.16)极小值的问题。由于 $\hat{f}$ 除了要满足 $J(\hat{f})$ 为极小值以外没有其他约束条件，所以称这种复原方法为无约束复原方法。式(9.16)的极小值可以用一般的极值求解方法求得。令

$$\frac{\partial J(\hat{f})}{\partial \hat{f}} = -2H^{\mathrm{T}}(g - H\hat{f}) = 0 \tag{9.17}$$

则

$$\hat{f} = (H^{\mathrm{T}}H)^{-1} \cdot H^{\mathrm{T}}g \tag{9.18}$$

其中，$(H^{\mathrm{T}}H)^{-1} \cdot H^{\mathrm{T}}$ 为矩阵 $H$ 的广义逆。当 $H^{-1}$ 存在时，式(9.18)可以简化为

$$\hat{f} = H^{-1}g \tag{9.19}$$

式(9.19)即为无约束条件下的复原方程解，虽然其形式较为简单，但是计算量是非常大的，需要对 $H^{-1}$ 进行简化。

### 2. 有约束复原方程

根据不同应用领域的需要，有时会对 $\hat{f}$ 设置一些约束条件，使处理得到的图像满足某些条件。在这种约束条件下求解方程(9.16)需要使用拉格朗日乘数法。

假设设置约束条件为复原前、后的图像具有相同的能量，则此约束条件的数学表示形式如下：

$$\hat{f}^{\mathrm{T}}\hat{f} = g^{\mathrm{T}}g = C \tag{9.20}$$

现在要求函数 $J(\hat{f})$ 在满足条件(9.20)条件下的极值。使用拉格朗日乘法，引入拉格朗日乘子 $\lambda$：

$$J(\hat{f},\lambda) = J(\hat{f}) + \lambda(\hat{f}^{\mathrm{T}}\hat{f} - C) = (g - H\hat{f})^{\mathrm{T}} \cdot (g - H\hat{f}) + \lambda(\hat{f}^{\mathrm{T}}\hat{f} - C) \tag{9.21}$$

令

$$\frac{\partial J(\hat{f},\lambda)}{\partial \hat{f}} = -2H^{\mathrm{T}}(g - H\hat{f}) + 2\lambda\hat{f} = 0 \tag{9.22}$$

则

$$\hat{f} = (H^{\mathrm{T}}H + \lambda I)^{-1} \cdot H^{\mathrm{T}}g \tag{9.23}$$

式中 $I$ 表示单位矩阵。式(9.23)就是图像在能量保持条件下得到的估计值表达式。一般情况下可以这样来描述有约束条件下的复原方程：假设 $Q$ 是 $F$ 的线性算子，约束复原问题就是求函数 $\|Q\hat{f}\|^2$ 在约束条件

$$\|g - H\hat{f}\|^2 = \|n\|^2 \tag{9.24}$$

下的极值问题。采用拉格朗日乘法对该问题进行求解，最后可得：

$$\hat{f} = \left(H^{\mathrm{T}}H + \frac{1}{\lambda}Q^{\mathrm{T}}Q\right)^{-1} \cdot H^{\mathrm{T}}g \tag{9.25}$$

式(9.25)就是有约束复原方程的一般公式。在实际运算过程中需要不断地调整参数 $\lambda$ 的取值，直至得到满意的效果。

### 9.3.2 分块循环矩阵的对角化

分块循环矩阵 $H$ 的对角化能够简化估计值的求解。$H$ 是一个 $MN \times MN$ 的矩阵，由 $M^2$ 个循环子矩阵组成。可以使用一个变换矩阵对其进行变换，最终得到一个对角矩阵。

令 $W$ 为变换矩阵，$W$ 的大小也是 $MN \times MN$，包含 $M^2$ 个子矩阵。$W$ 的第 $im$ 个子矩阵的定义为

$$W(i,m) = w_M(i,m)w_N(k,n) \tag{9.26}$$

其中，$i,m=0, 1, 2, \cdots, M-1$；$k,n=0, 1, 2, \cdots, N-1$

$$w_M(i,m) = \exp\left[\mathrm{j}\,\frac{2\pi}{M}im\right]$$

$$w_N(k,n) = \exp\left[\mathrm{j}\,\frac{2\pi}{M}kn\right]$$

矩阵 $W$ 的逆矩阵 $W^{-1}$ 的第 $im$ 个子矩阵的定义如下：

$$W^{-1}(i,m) = w_M^{-1}(i,m)w_N^{-1}(k,n) \tag{9.27}$$

其中，$i,m=0, 1, 2, \cdots, M-1$；$k,n=0, 1, 2, \cdots, N-1$

$$w_W^{-1}(i,m) = \exp\left[-\mathrm{j}\,\frac{2\pi}{M}im\right]$$

$$w_N^{-1}(k,n) = \exp\left[-\mathrm{j}\,\frac{2\pi}{M}kn\right]$$

可以证明，分块循环矩阵 $H$ 能够写成如下形式：

$$H = WDW^{-1} \tag{9.28}$$

其中，$D$ 是对称正交对角阵，其具体取值如下：

$$D(k,k) = \sum_{i=0}^{M-1} h(i) \exp\left[-\mathrm{j}\,\frac{2\pi ki}{M}\right] \qquad k = 0,\ 1,\ 2,\ \cdots,\ M-1$$

于是可以得到

$$\boldsymbol{W}^{-1}\boldsymbol{g} = \boldsymbol{W}^{-1}(\boldsymbol{W}\boldsymbol{D}\boldsymbol{W}^{-1})\boldsymbol{f} + \boldsymbol{W}^{-1}\boldsymbol{n} = \boldsymbol{D}\boldsymbol{W}^{-1}\boldsymbol{f} + \boldsymbol{W}^{-1}\boldsymbol{n} \tag{9.29}$$

根据 $\boldsymbol{W}^{-1}$的表达式可知：

$$\boldsymbol{W}^{-1}\boldsymbol{g} = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} \boldsymbol{g}(x,y) \exp\left[-\mathrm{j}2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)\right]$$

其中 $u=0,\ 1,\ 2,\ \cdots,\ M-1$；$v=0,\ 1,\ 2,\ \cdots,\ N-1$。

由上式可见，$\boldsymbol{W}^{-1}\boldsymbol{g}$ 实际上就是由矩阵 $\boldsymbol{g}$ 各个分量的傅立叶变换按行堆叠而成的。同理，$\boldsymbol{W}^{-1}\boldsymbol{f}$ 和 $\boldsymbol{W}^{-1}\boldsymbol{n}$ 分别是矩阵 $\boldsymbol{f}$ 和 $\boldsymbol{n}$ 各个分量的傅立叶变换按行堆叠而成的向量。这样，我们就可以将空域中复杂的方程求解问题转化到频域中进行简单计算了。

### 9.3.3 无约束复原方程求解

对无约束复原方程而言，所谓退化模型求解就是要求解以下方程：

$$\hat{\boldsymbol{f}} = \boldsymbol{H}^{-1}\boldsymbol{g} = \boldsymbol{W}\boldsymbol{D}^{-1}\boldsymbol{W}^{-1}\boldsymbol{g} \tag{9.30}$$

因为

$$H(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} h(x,y) \exp\left[-\mathrm{j}2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)\right]$$

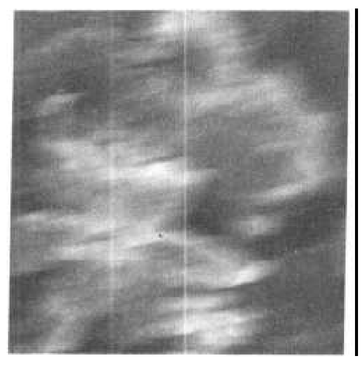
根据以上的分块循环矩阵的对角化过程可知，如果假设 $\boldsymbol{g}$、$\boldsymbol{f}$ 和 $\boldsymbol{n}$ 的傅立叶变换分别为 $G$、$F$ 和 $N$，那么式(9.29)可以写为

$$G(u,v) = MNH(u,v)F(u,v) + N(u,v) \tag{9.31}$$

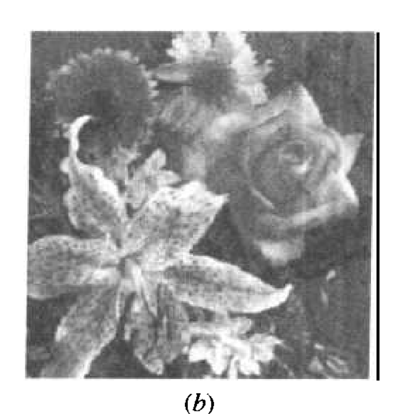
由式(9.29)、(9.30)和(9.31)可得

$$\boldsymbol{W}^{-1}\hat{\boldsymbol{f}} = \hat{F}(u,v) = \frac{G(u,v)}{H(u,v)} \tag{9.32}$$

由式(9.32)可以看出，如果首先求出 $\boldsymbol{g}$ 和 $\boldsymbol{h}$ 相应离散傅立叶变换的比值，然后再对其进行傅立叶反变换就可以获得需要的方程解，也就是复原后的图像函数。通过以上的讨论可知，通过对分块循环矩阵进行对角化，可以将空域中庞大的 $MN\times MN$ 的矩阵求解问题转化为频域中的 $M\times N$ 次傅立叶变换求解，这不但能够提高图像的复原速度，还可以节约大量的计算机内存。图 9.3 给出了一幅无噪声的模糊图像及其复原效果。







(a) (b)

图 9.3 无约束图像复原前、后的显示效果比较

(a) 复原前；(b) 复原后

### 9.3.4 最小二乘方滤波复原

最小二乘方滤波复原方法主要是针对有约束退化模型而言的。由式(9.25)可知，对于一个一般的有约束方程，估计值 $\hat{f}$ 满足以下方程：

$$\hat{f}=\left(H^{T}H+\frac{1}{\lambda}Q^{T}Q\right)^{-1}H^{T}g$$

定义：$Q^{T}Q=R_{f}^{-1}R_{n}$，其中 $R_f$ 是 $f$ 的自相关矩阵，$R_n$ 是 $n$ 的自相关矩阵，分别定义如下：

$$R_{f}=E(f\cdot f^{T})$$
$$R_{n}=E(n\cdot n^{T})$$

这里，$E$ 表示数学期望。可以证明 $R_f$ 和 $R_n$ 都可以近似为分块循环矩阵，因此可以使用以上介绍的方法对它们进行对角化，于是有

$$\left.\begin{aligned}R_{f}&=WAW^{-1}\\R_{n}&=WBW^{-1}\end{aligned}\right\}\tag{9.33}$$

其中，$A$ 和 $B$ 为对角阵，对角元素分别为 $R_f$ 和 $R_n$ 中相应元素的傅立叶变换。一般将信号自相关函数的离散傅立叶变换称为信号的功率谱密度，因此又称 $A$ 和 $B$ 为 $f(x,y)$ 和 $n(x,y)$ 的功率谱密度，分别用 $S_f(u,v)$ 和 $S_n(u,v)$ 表示。将 $R_f$ 和 $R_n$ 代入式(9.29)得：

$$\hat{f}=\left(WD^{T}DW^{-1}+\frac{1}{\lambda}WA^{-1}BW^{-1}\right)^{-1}WD^{T}W^{-1}g\tag{9.34}$$

将上式两边同时乘以 $W^{-1}$ 并化简得：

$$W^{-1}\hat{f}=\left(D^{T}D+\frac{1}{\lambda}A^{-1}B\right)^{-1}D^{T}W^{-1}g\tag{9.35}$$

根据以上讨论过的概念将上式表示为傅立叶变换的形式

$$\hat{F}(u,v)=\left[\frac{H^{T}(u,v)}{|H(u,v)|^{2}+1/\lambda[S_{n}(u,v)/S_{f}(u,v)]}\right]G(u,v)\tag{9.36}$$

因为

$$|H(u,v)|^{2}=H^{T}(u,v)H(u,v)$$

因而式(9.36)可以写为

$$\hat{F}(u,v)=\left[\frac{1}{H(u,v)}\cdot\frac{|H(u,v)|^{2}}{|H(u,v)|^{2}+1/\lambda[S_{n}(u,v)/S_{f}(u,v)]}\right]G(u,v)\tag{9.37}$$

分析式(9.37)可以看出，当 $S_n(u,v)=0$ 时，式(9.37)与无约束复原方程解的形式是一致的，称之为反向滤波器表示式；当 $\lambda=1$ 时，称式(9.37)为魏纳滤波器表示式。可以证明，此时得到的估计值是使 $E\{[f(x,y)-\hat{f}(x,y)]^{2}\}$ 取最小值的最优估计值，其所形成的滤波器也称为最小无误差滤波器；当 $\lambda$ 是一个不为零的变量时，称式(9.37)为参变魏纳滤波器表示式。

对于魏纳滤波器，当 $S_f(u,v)$ 和 $S_n(u,v)$ 未知时，常常使用比例系数 $K$ 来表示 $S_n(u,v)$ 与 $S_f(u,v)$ 的比值，这样就可以将式(9.37)近似为

$$\hat{F}(u,v)\approx\left[\frac{1}{H(u,v)}\cdot\frac{|H(u,v)|^{2}}{|H(u,v)|^{2}+K}\right]G(u,v)\tag{9.38}$$

在实际应用中就是使用(9.38)式进行图像复原操作的。

将参变魏纳滤波器用于图像复原也称为约束最小二乘方复原，这种图像复原方法只需有关噪声均值和方差的知识就能够对每一幅给定的退化图像进行复原，以得到最优的效果。约束二乘方复原方法也是从式(9.25)出发的，其基本问题还是要确定矩阵 $\boldsymbol{Q}$。由于式(9.25)实际上是一个病态方程，有时该方程的解将振荡得非常厉害。为了减小振荡，可以建立一种基于平滑测度的最优准则进行复原，例如，使某些二阶微分函数最小化等。函数 $f(x,y)$ 在 $(x,y)$ 处的二阶微分可以用下式来近似表示：

$$\frac{\partial^2 f}{\partial x^2}+\frac{\partial^2 f}{\partial y^2}=4f(x,y)-[f(x+1,y)+f(x-1,y)+f(x,y+1)+f(x,y-1)] \tag{9.39}$$

上述微分可以使用以下拉氏算子与 $f(x,y)$ 进行卷积得到

$$C_L(x,y)=\begin{bmatrix}0 & -1 & 0\\ -1 & 4 & -1\\ 0 & -1 & 0\end{bmatrix} \tag{9.40}$$

为了避免卷积周期重叠，需要将 $C_L(x,y)$ 的周期扩充为 $f(x,y)$ 的周期。

经常使用的一个基于二阶微分的最优准则表达式如下：

$$\min\left\{\left(\frac{\partial^2 f}{\partial x^2}+\frac{\partial^2 f}{\partial y^2}\right)^2\right\} \tag{9.41}$$

式(9.41)的矩阵形式如下：

$$\min\{\boldsymbol{f}^{\mathrm{T}}\boldsymbol{C}^{\mathrm{T}}\boldsymbol{C}\boldsymbol{f}\} \tag{9.42}$$

其中

$$\boldsymbol{C}=\begin{bmatrix}C_0 & C_{M-1} & C_{M-2} & \cdots & C_1\\ C_1 & C_0 & C_{M-1} & \cdots & C_2\\ C_2 & C_1 & C_0 & \cdots & C_3\\ \cdots & \cdots & \cdots & \cdots & \cdots\\ C_{M-1} & C_{M-2} & C_{M-3} & \cdots & C_0\end{bmatrix}$$

$\boldsymbol{C}$ 的每一个子矩阵 $C_j$ 都是由 $C_L(x,y)$ 的第 $j$ 行元素组成的矩阵

$$\boldsymbol{C}_j=\begin{bmatrix}C_L(j,0) & C_L(j,N-1) & C_L(j,N-2) & \cdots & C_L(j,1)\\ C_L(j,1) & C_L(j,0) & C_L(j,N-1) & \cdots & C_L(j,2)\\ C_L(j,2) & C_L(j,1) & C_L(j,0) & \cdots & C_L(j,3)\\ \cdots & \cdots & \cdots & \cdots & \cdots\\ C_L(j,N-1) & C_L(j,N-2) & C_L(j,N-3) & \cdots & C_L(j,0)\end{bmatrix}$$

(9.42)式中的 $\boldsymbol{C}$ 称为平滑矩阵，由 $C_L$ 的周期性可知，$\boldsymbol{C}$ 是一个分块循环矩阵，于是对 $\boldsymbol{C}$ 进行对角化

$$\boldsymbol{C}=\boldsymbol{W}\boldsymbol{E}\boldsymbol{W}^{-1} \tag{9.43}$$

其中，$\boldsymbol{E}$ 是一个对角阵，对角元素为 $C_L(x,y)$ 相应元素的傅立叶变换。如果令 $\boldsymbol{Q}=\boldsymbol{C}$，那么

$$\hat{\boldsymbol{f}}=\left(\boldsymbol{H}^{\mathrm{T}}\boldsymbol{H}+\frac{1}{\lambda}\boldsymbol{C}^{\mathrm{T}}\boldsymbol{C}\right)^{-1}\boldsymbol{H}^{\mathrm{T}}\boldsymbol{g} \tag{9.44}$$

于是可以得到

$$\hat{\boldsymbol{f}}=\left(\boldsymbol{W}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{W}^{-1}+\frac{1}{\lambda}\boldsymbol{W}\boldsymbol{E}^{\mathrm{T}}\boldsymbol{E}\boldsymbol{W}^{-1}\right)^{-1}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{W}^{-1}\boldsymbol{g} \tag{9.45}$$

将式(9.45)两边同乘以 $W^{-1}$ 并化简可得：

$$W^{-1}\hat{f}=\left(D^{T}D+\frac{1}{\lambda}E^{T}E\right)^{-1}D^{T}W^{-1}g \tag{9.46}$$

故

$$\hat{F}(u,v)=\left[\frac{1}{H(u,v)}\cdot\frac{|H(u,v)|^{2}}{|H(u,v)|^{2}+1/\lambda|C_{L}(u,v)|^{2}}\right]G(u,v) \tag{9.47}$$

显然，式(9.47)就是一个参变魏纳滤波器的形式。对于这个参变魏纳滤波器，最突出的问题就是要调整 $\lambda$ 的值，使其满足约束条件(9.24)式。只有 $\lambda$ 满足这一条件，才能够根据式(9.47)求出最优复原解。$\lambda$ 可以使用迭代方法来确定，迭代过程如下：

定义剩余向量 $R$ 为

$$R=g-H\hat{f} \tag{9.48}$$

将式(9.44)代入此式得：

$$R=g-H\left(H^{T}H+\frac{1}{\lambda}C^{T}C\right)^{-1}H^{T}g \tag{9.49}$$

令 $r=1/\lambda$，分析可知，$\|R\|^{2}$ 是 $r$ 的单调函数，因此力求调整 $r$ 使其满足：

$$\|R\|^{2}\leqslant\|n\|^{2}\pm\alpha \tag{9.50}$$

其中，$\alpha$ 为准确度因子，$\alpha$ 数值越小越符合约束条件的要求。从以上的分析可以看出，如果想确定 $r$，那么必须了解 $\|n\|^{2}$ 的情况。$\|n\|^{2}$ 可以由噪声的平均值和方差计算得出。假设噪声的方差为

$$\sigma_{n}^{2}=E\{[n(x,y)-\bar{n}]^{2}\}=E\{n^{2}(x,y)-\bar{n}^{2}\} \tag{9.51}$$

其中

$$\bar{n}=\frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}n(x,y)$$

是 $n(x,y)$ 的平均值。如果使用采样平均值来近似 $n^{2}(x,y)$ 的期望值，那么式(9.51)变为

$$\begin{aligned}\sigma_{n}^{2}&=\frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}n^{2}(x,y)-\bar{n}^{2}\\&=\frac{\|n\|^{2}}{MN}-\bar{n}^{2}\end{aligned} \tag{9.52}$$

于是可知

$$\|n\|^{2}=MN(\sigma_{n}^{2}+\bar{n}^{2}) \tag{9.53}$$

通过以上的分析可知，约束最小二乘方复原的整个过程如下：

(1) 给定 $\alpha$ 以及 $r$ 增减的步长，并选择 $r$ 的初始值，利用式(9.53)估计 $\|n\|^{2}$。

(2) 利用式(9.47)求出 $\hat{F}(u,v)$，计算其傅立叶反变换，得到 $\hat{f}$。

(3) 根据式(9.48)计算 $R$ 以及 $\|R\|^{2}$。

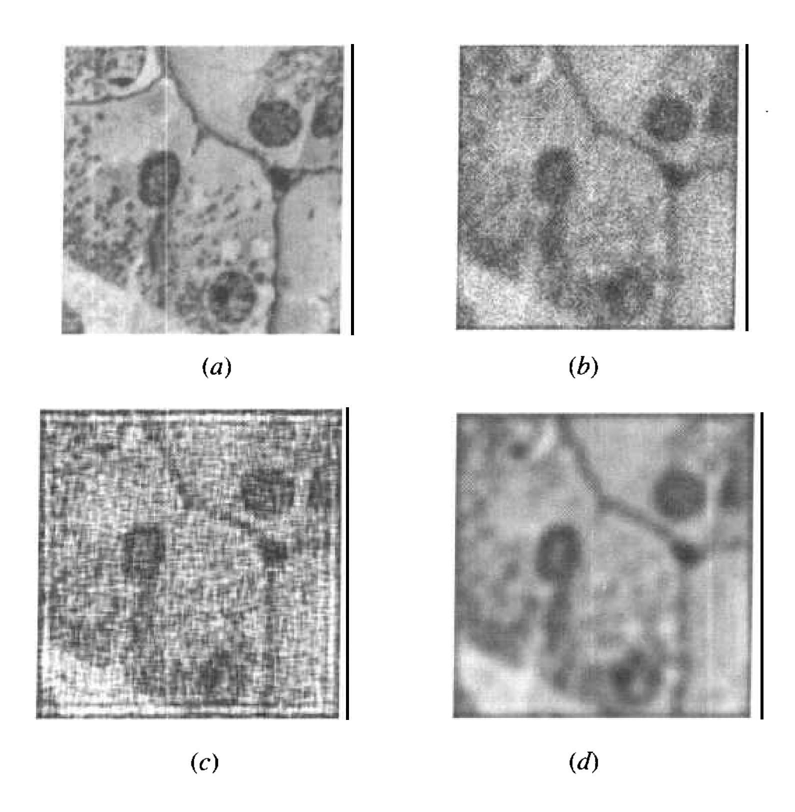
(4) 如果 $\|R\|^{2}<\|n\|^{2}-\alpha$，那么 $r$ 增加一个步长；如果 $\|R\|^{2}>\|n\|^{2}-\alpha$，那么 $r$ 减小一个步长。

(5) 如果 $\|R\|^{2}\leqslant\|n\|^{2}\pm\alpha$，那么复原过程结束，根据此时的 $r$ 值计算得出的 $\hat{f}$ 就是复原后的图像；否则继续进行第(2)步。

例如，图 9.4(*d*)是对图 9.4(*b*)所示的图像(由图 9.4(*a*)模糊并添加噪声得来)采用约

束最小二乘方复原的结果，而图 9.4(*c*)则是使用魏纳滤波器得到的复原结果。由此可见，在噪声比较明显的情况下，采用约束最小二乘方复原的结果与魏纳滤波的效果相比有十分明显的改进。但是当图像中只有模糊而不存在噪声时，两种复原方法的效果是相差无几的。

除了以上介绍的复原方法之外，针对复原方程的病态性，还提出了很多算法，例如几何平均滤波器、最大熵滤波器等滤波器算法，这里我们就不一一介绍了，读者可以参考有关书籍了解这些算法的原理和使用。



(*a*)　(*b*)

(*c*)　(*d*)

图 9.4　有噪声模糊图像的魏纳滤波与约束最小二乘方滤波后显示效果比较
(*a*) 原图像；(*b*) 模糊并添加噪声后；
(*c*) 对(*b*)进行魏纳滤波后；(*d*) 对(*b*)进行约束最小二乘方滤波后

## 9.4　图像复原的 MATLAB 实现方法

### 9.4.1　模糊及噪声

为了说明 MATLAB 图像复原函数的效果，针对每一个例子中的复原或噪声图像我们都给出了其原始图像 f，用来表示在图像获取状态完美无缺的情况下所应该具有的性质(f 实际上并不存在)。这样就可以将复原后的图像与原始图像相比较，从而看出复原的效果如何。事实上例子中给出的模糊或噪声图像都是通过对原始图像人为添加运动模糊和各种噪声而形成的。基于以上的原因，这里首先对 MATLAB 图像模糊化和添加噪声的函数作以介绍。

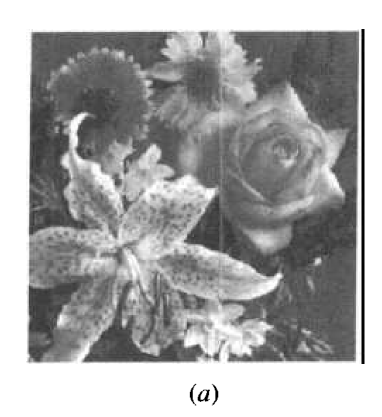
根据以上两节的分析可知，如果图像中不存在噪声，那么其模糊状况完全是由 PSF 决定的。在这种情况下，去模糊的基本任务就是使用精确描述失真的 PSF 对模糊图像进行去卷积操作。为了创建模糊化的图像，通常使用 MATLAB 的图像处理工具箱函数 fspecial 创建一个确定类型的 PSF，然后使用这个 PSF 对原始图像进行卷积，从而得到模糊化的图像。fspecial 函数的基本调用格式在上一章中已经作过介绍，此处使用两个实例来说明如何使用该函数来模糊一幅图像。

**例 9.1**：创建一个仿真运动模糊的 PSF 来模糊如图 9.5(*a*)所示的图像，指定运动位移为 31 个像素，运动角度为 11°。

首先要使用 fspecial 函数创建 PSF，然后调用 imfilter 函数使用 PSF 对原始图像 I 进行卷积，这就可以得到一幅模糊图像 Blurred。

```
I = imread('flowers.tif');
I = I(10+[1:256],222+[1:256],:);        % 剪裁图像
subplot(1,2,1);imshow(I);
LEN = 31;
THETA = 11;
PSF = fspecial('motion',LEN,THETA);
Blurred = imfilter(I,PSF,'circular','conv');
subplot(1,2,2); imshow(Blurred);
```

模糊化的图像如图 9.5(*b*)所示。



(*a*)



(*b*)

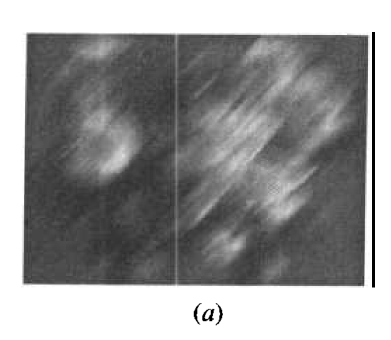
图 9.5　模糊化前、后图像显示效果比较

(*a*) 模糊前；(*b*) 模糊后

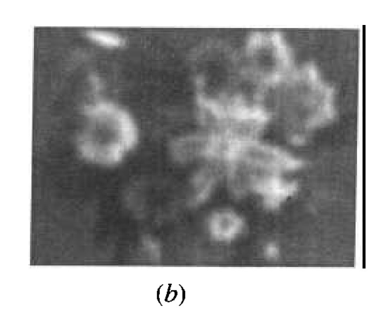
**例 9.2**：对图 9.5(*a*)所示的图像分别采用运动 PSF 和均值滤波 PSF 进行模糊，观察不同的 PSF 产生的效果。

```
I = imread('flowers.tif');
H = fspecial('motion',50,45);           %运动 PSF
MotionBlur = imfilter(I,H);
subplot(1,2,1);imshow(MotionBlur);
H = fspecial('disk',10);                %均值滤波 PSF
blurred = imfilter(I,H);
subplot(1,2,2);imshow(blurred);
```

两种 PSF 产生的不同模糊化图像分别如图 9.6(*a*)和 9.6(*b*)所示。



(*a*)



(*b*)

图 9.6　运动 PSF 和均值滤波 PSF 产生的模糊图像效果比较

(*a*) 运动 PSF 的模糊图像；(*b*) 均值滤波 PSF 的模糊图像

一般在需要复原的图像中不但包含模糊成分，而且还有一些额外的噪声成分。在 MATLAB 中可以使用两种方法模拟图像噪声：一种是使用 imnoise 函数直接对图像添加固定类型的噪声；另一种是创建自定义的噪声，然后使用 MATLAB 图像代数运算函数imadd 将其添加到图像中去。这两种方法中用到的函数在前面的章节中已经作过介绍，此处不再赘述。以下给出两个例子说明这两种方法的具体操作。

**例 9.3**：给图 9.5(*a*)所示的图像添加均值为 0，方差为 0.02 的高斯噪声。

对于高斯噪声、泊松噪声、椒盐噪声等 MATLAB 系统预定义的噪声类型来说，使用 imnoise 函数将其添加到图像中是 MATLAB 噪声模拟最简单的方法。对于本例使用的程序代码如下：

```
I = imread('flowers.tif');
I = I(10+[1:256],222+[1:256],:);
V = .02;
Noisy = imnoise(I,'gaussian',0,V);
figure;imshow(Noisy);
```

添加了高斯噪声后的图像如图 9.7 所示。

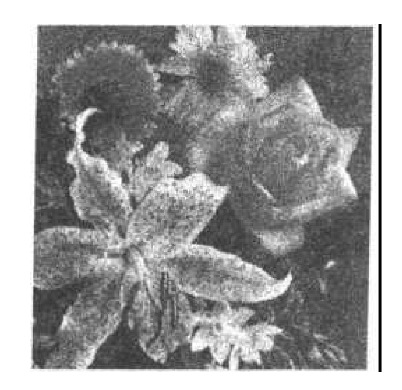


图 9.7　添加高斯噪声后的图像效果

**例 9.4**：给图 9.5(*a*)所示的图像添加随机噪声。

由于 imnoise 函数不提供随机噪声添加功能，所以使用以上介绍的第二种方法来完成所需工作。程序代码如下：

```
I = imread('flowers.tif');
I = I(10+[1:256],222+[1:256],:);
noise = 0.1 * randn(size(I));
Noisy = imadd(I,im2uint8(noise));
figure;imshow(Noisy);
```

添加了随机噪声后的图像如图 9.8 所示。



图 9.8　添加随机噪声后的图像效果

### 9.4.2 MATLAB 复原函数简介

MATLAB 的图像处理工具箱包含四个图像复原函数，按照这些函数的复杂程度将其排列如下：

- deconvwnr 函数：使用魏纳滤波复原；
- deconvreg 函数：使用约束最小二乘方滤波复原；
- deconvlucy 函数：使用 Lucy-Richardson 复原；
- deconvblind 函数：使用盲去卷积算法复原。

以上所有复原函数都是以一个 PSF 和模糊图像作为主要输入参数的。deconvwnr 函数可以实现最小均方误差复原，而函数 deconvreg 可以实现约束最小均方误差复原，可以在这种复原方法中对输出图像采用某些约束(缺省情况下为光滑性约束)。无论是哪一个函数在图像复原过程中，用户都应该向其提供有关噪声的信息。

deconvlucy 函数可以实现一个加速收敛的 Lucy-Richardson 算法，这个函数将使用最优化技术和泊松统计完成多次反复过程，该函数无需模糊图像的噪声信息。deconvblind 函数实现盲去卷积算法，在执行过程中可以不需要有关PSF 的知识，调用时将PSF 的一个初始化估计作为输入参数即可。deconvblind 函数将给复原后的图像返回一个重建的 PSF。该实现过程使用与 deconvlucy 函数相同的收敛方式。

除了以上四个复原函数以外，还可以使用 MATLAB 自定义的复原函数。

**注意：**

*用户可能需要执行多次复原过程，每一次反复过程中指定的复原函数参数都要发生变化，直至根据有限的先验知识认为图像已经达到了近似真实场景最好的效果为止。在这个过程中，用户必须进行多次判断，决定新出现的特征是原始场景的特征还是由复原过程导致的人工痕迹。*

**小技巧：**

*由于图像的边界对图像而言是不连续的，所以复原操作常常会产生“环”。为了避免这种现象，可以在复原之前使用 edgetaper 函数对图像进行预处理。edgetaperj 函数可以用来消除图像的不连续边界。*

### 9.4.3 魏纳滤波复原

通过调用 deconvwnr 函数可以利用魏纳滤波方法对图像进行复原处理。当图像的频率特性和噪声已知(至少部分已知)时，魏纳滤波的效果非常好。deconvwnr 函数的调用格式如下：

```
J = DECONVWNR(I,PSF,NCORR,ICORR)
```

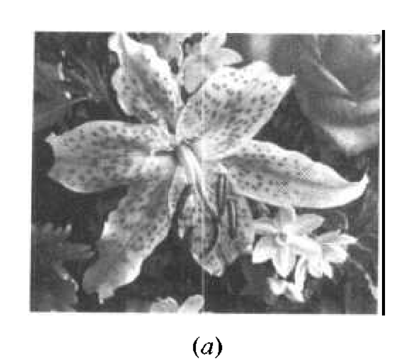
或

```
J = DECONVWNR(I,PSF,NSR)
```

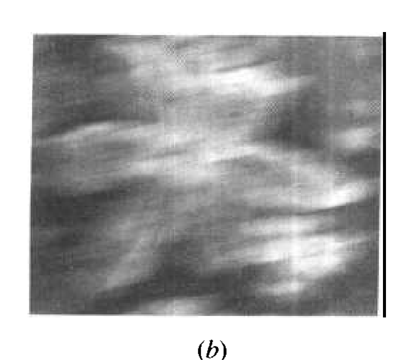
其中，I 表示输入图像，PSF 表示点扩散函数，NSR(缺省值为 0)、NCORR 和 ICORR 都是可选参数，分别表示信噪比、噪声的自相关函数、原始图像的自相关函数。输出参数J 表示复原后的图像。下面给出两个例子说明使用 deconvwnr 函数进行图像复原的具体实现方

法，读者可以从中体会各个参数的用途。

**例 9.5**：使用函数 deconvwnr 对图 9.9(*b*)所示的无噪声模糊图像进行复原重建，观察所得结果，并与原始图像(如图 9.9(*a*)所示)进行比较。



(*a*)



(*b*)

图 9.9　图像及其无噪声模糊图像

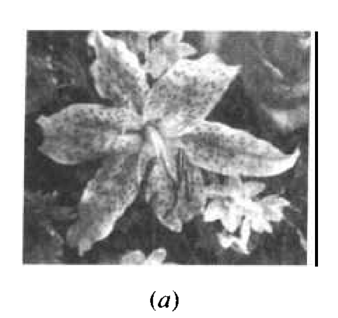
(*a*) 原始图像；(*b*) 无噪声模糊图像

在对图 9.9(*b*)所示的无噪声模糊图像进行复原时，首先假设真实的 PSF 是已知的，读入图像后使用以下程序代码实现图像复原：

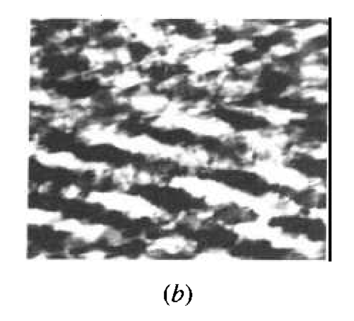
```
...              %读入原始图像
LEN = 31;
THETA = 11;
PSF = fspecial('motion',LEN,THETA);
wnr1 = deconvwnr(Blurred,PSF);
figure;imshow(wnr1);
```

复原结果如图 9.10(*a*)所示。在实际应用过程中，真实的 PSF 通常是未知的，需根据一定的先验知识对 PSF 进行估计，再将估计值作为参数进行图像复原。图9.10(*b*)和 9.10(*c*)分别显示了使用较“长”和较“陡峭”的 PSF 后所产生的复原效果，将该结果与图 9.10(*a*)进行对比就可以体会到 PSF 的重要性。其程序代码如下：

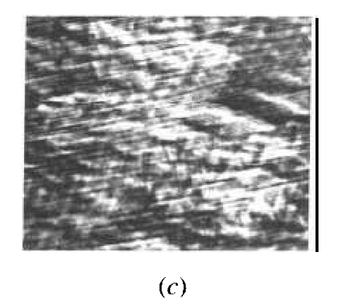
```
wnr2 = deconvwnr(Blurred,fspecial('motion',2 * LEN,THETA));      %长 PSF
subplot(1,2,1);imshow(wnr2);
wnr3 = deconvwnr(Blurred,fspecial('motion',LEN,2 * THETA));      %陡峭 PSF
subplot(1,2,2);imshow(wnr3);
```



(*a*)



(*b*)



(*c*)

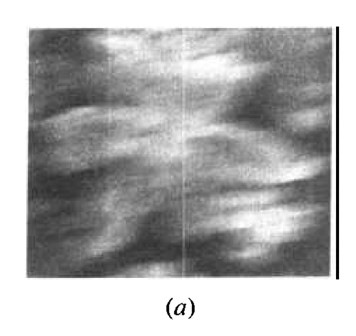
图 9.10　不同 PSF 产生的复原效果比较

(*a*) 使用真实的 PSF 复原；(*b*) 使用较“长”的 PSF 复原；(*c*) 使用较“陡峭”的 PSF 复原

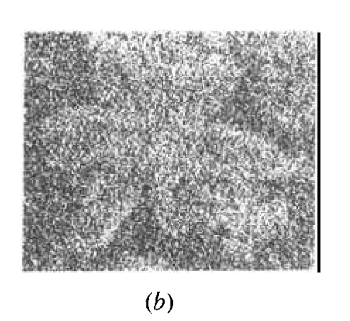
**例 9.6**：对图 9.11(*a*) 的有噪声模糊图像(原始图像见图 9.9(*a*))进行恢复并尽量提高重建质量。

如果直接使用以下代码调用 deconvwnr 函数对一幅有噪声的图像进行复原(结果如图 9.11(*b*)所示)，那么复原效果是很不好的，这是由魏纳滤波器的所导致的。

```
...          %读入图像 9.11(a)，图像矩阵名为 I
wnr4 = deconvwnr(I,PSF);
subplot(2,2,2);imshow(wnr4);
```



(*a*)



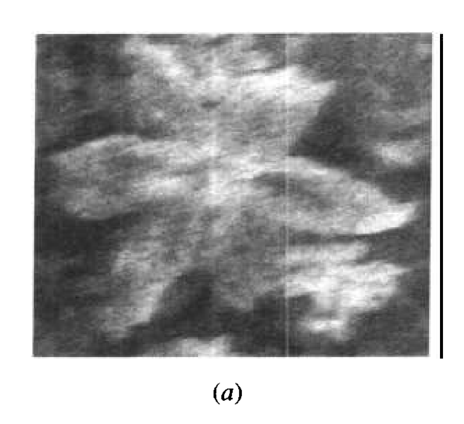
(*b*)

图 9.11　有噪声模糊图像及其直接魏纳滤波后的显示效果对比
(*a*) 有噪声模糊图像；(*b*) 直接魏纳滤波后

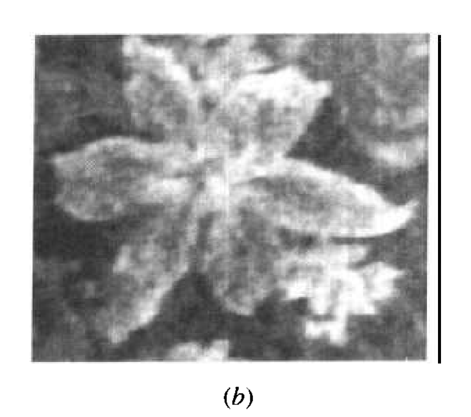
为了提高重建图像的质量，MATLAB 使用三个可选的参数：NSR、NCORR 和 ICORR，这三个参数的使用都需要一定的噪声信息。假设图 9.11(*b*)中的噪声是由以下方法产生的：

```
noise = 0.1 * randn(size(I));
...        %模糊化图像并放入矩阵 Blurred 中
BlurredNoisy = imadd(Blurred,im2uint8(noise));
```

以下代码将分别使用这三个参数对图像进行复原，复原的效果如图 9.12(*a*)和 9.12(*b*)所示。显然，使用这些可选参数得到的图像质量较图 9.11(*b*)而言有了较大的提高，读者可以通过修改这些参数来观察参数对图像复原效果的影响。



(*a*)



(*b*)

图 9.12　使用可选参数进行图像复原后的显示效果对比
(*a*) 设置信噪比参数后；(*b*) 设置噪声和图像自相关函数后

```
NSR = sum(noise(:).^2)/sum(im2double(I(:)).^2);
wnr5 = deconvwnr(BlurredNoisy,PSF,NSR);
subplot(1,2,1);imshow(wnr5);
NP = abs(fftn(noise)).^2;
NCORR = fftshift(real(ifftn(NP)));            %噪声自相关函数
IP = abs(fftn(im2double(I))).^2;
ICORR = fftshift(real(ifftn(IP)));            %图像自相关函数
wnr6 = deconvwnr(BlurredNoisy,PSF,NCORR,ICORR);
subplot(1,2,2);imshow(wnr6);
```

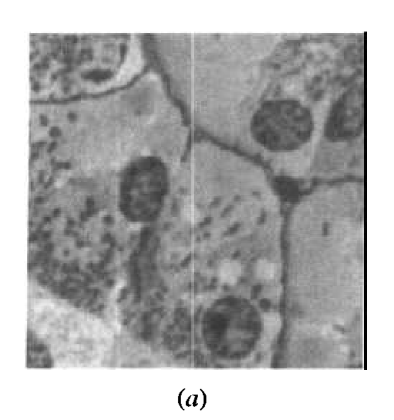
### 9.4.4 约束最小二乘方滤波复原

使用 deconvreg 函数可以利用约束最小二乘方滤波对图像进行复原。约束最小二乘方滤波方法可以在噪声信号所知有限的条件下很好地工作。deconvreg 函数的调用格式如下：

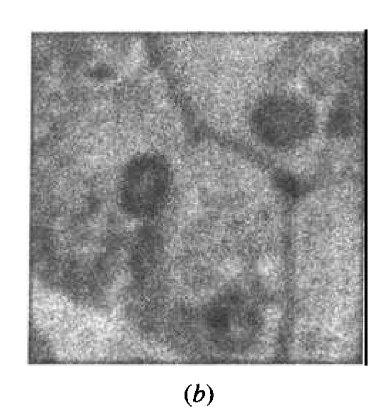
```
[J LRANGE]= DECONVREG(I,PSF,NP,LRANGE,REGOP)
```

其中，I 表示输入图像。PSF 表示点扩散函数。NP、LRANGE(输入)和 REGOP 是可选参数，分别表示图像的噪声强度、拉氏算子的搜索范围(该函数可以在指定的范围内搜索最优的拉氏算子)和约束算子，这三个参数的缺省值分别为：0、$[10^{-9}, 10^{9}]$和平滑约束拉氏算子。返回值 J 表示复原后的输出图像。返回值 LRANGE 表示函数执行时最终使用的拉氏算子。下面给出一个例子来说明约束最小二乘方复原方法的实现过程。

**例 9.7**：对图 9.13(*b*)给出的有噪声模糊图像(其原始图像如图 9.13(*a*)所示)使用最小二乘方滤波方法进行复原重建，要求尽量提高重建图像的质量。



(*a*)



(*b*)

图 9.13　原始图像及其有噪声模糊化图像

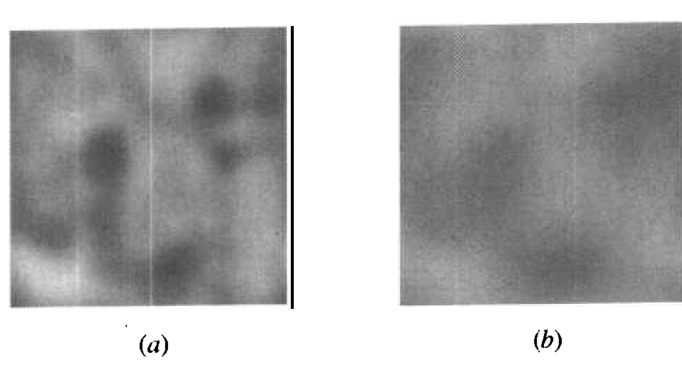
(*a*) 原始图像；(*b*) 有噪声模糊图像

首先使用以下代码说明以上介绍的参数的使用方法。

```
...%读入模糊图像并命名为 BlurredNoisy
V = .02;
NP = V * prod(size(I));                        % 计算噪声强度
Edged = edgetaper(BlurredNoisy,PSF);           %图像预处理
[reg1 LAGRA] = deconvreg(Edged,PSF,NP);
subplot(1,2,1),imshow(reg1),title('Restored with NP');
reg2 = deconvreg(Edged,PSF,NP * 1.2);
```

```
subplot(1,2,2);imshow(reg2);
reg3 = deconvreg(Edged,PSF,[],LAGRA);
figure;subplot(1,2,2);imshow(reg3);
reg4 = deconvreg(Edged,PSF,[],LAGRA * 50);
subplot(1,2,3);imshow(reg4);
REGOP = [1 -2 1];
reg5 = deconvreg(Edged,PSF,[],LAGRA,REGOP);
figure;imshow(reg5);
```

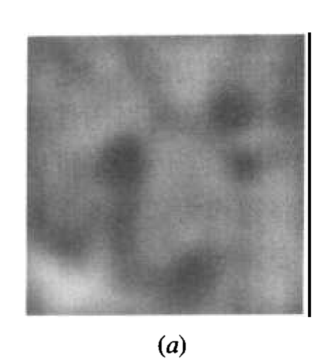
以上代码生成的复原图像分别如图 9.14、9.15 和图 9.16 所示，通过这些图像可以分析各个参数对图像复原质量的影响。在实际应用中，读者可以根据这些经验来选择最佳的参数进行图像复原。

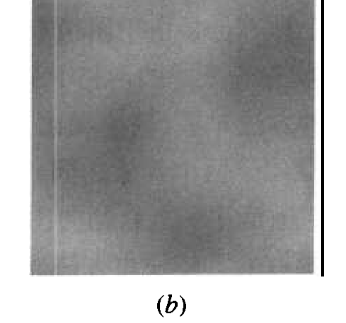


(*a*) (*b*)

图 9.14 不同信噪比复原结果比较

(*a*) 小 NP；(*b*) 大 NP





(*a*) (*b*)

图 9.15 不同拉氏算子搜索范围复原效果比较

(*a*) 小范围搜索；(*b*) 大范围搜索

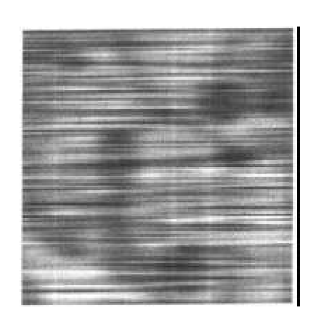


图 9.16 平滑约束复原效果

## 9.4.5 Lucy-Richardson 复原

使用 deconvlucy 函数，利用加速收敛的 Lucy-Richardson 算法可以对图像进行复原。Lucy-Richardson 算法能够求出按照泊松噪声统计标准求出与给定 PSF 卷积后最有可能成为输入模糊图像的图像。当 PSF 已知，但图像噪声信息未知时，可以使用这个函数进行有效的工作。deconvlucy 函数能够实现多种复杂图像重建算法，这些算法都是基于原始 Lucy - Richardson 最大化可能性算法的。deconvlucy 函数的调用格式如下：

```
J = DECONVLUCY(I,PSF,NUMIT,DAMPAR,WEIGHT,READOUT,SUBSMPL)
```

其中，I 表示输入图像。PSF 表示点扩散函数。其他参数都是可选参数：NUMIT 表示算法的重复次数，缺省值为 10；DAMPAR 表示偏差阈值，缺省值为 0(无偏差)；WEIGHT 表示像素记录能力，缺省值为原始图像的数值；READOUT 表示噪声矩阵，缺省值为 0；SUBSMPL 表示子采样时间，缺省值为 1。deconvlucy 函数的输出 J 是一个单元数组，包含四个元素：output{1}，原始输入图像；output{2}，最后一次反复产生的图像；output{3}，倒数第二次产生的图像；output{4}，deconvlucy 函数用来获知重新起始点的内部信息。输出的单元数组可以作为输入参数传递给 deconvlucy 函数，从而重新开始重复过程。

噪声痕迹是最大化可能性数据逼近算法的常见问题。经过多次重复处理后，尤其是在低信噪比条件下，重建图像可能会出现一些斑点，这些斑点并不代表图像的真实结构，只不过是输出图像过于逼近噪声所产生的结果。为了控制这些痕迹，deconvlucy 函数使用一个收敛参数 DAMPAR，这个参数指定收敛过程中结果图像与原始图像背离程度的阈值。对于那些超过阈值的数据，反复过程将被禁止。

图像重建的另一复杂之处在于那些可能包括坏像素的数据可能会随时间和位置的变化而变化。通过指定 deconvlucy 函数的 WEIGHT 数组参数，可以忽略图像中某些指定的像素。需要忽略的像素对应的 WEIGHT 数组元素值为 0。

CCD 检测器中的噪声有两个主要成分：一个是呈泊松分布的光子计算噪声；另一个是镜头呈高斯分布的读取噪声。Lucy-Richardson 算法的复原过程可以从根本上说明第一种类型的噪声，但是用户必须自己声明第二种噪声情况，deconvlucy 函数使用 READOUT 输入参数来指定这种噪声。READOUT 参数的值通常是读取噪声变量和背景噪声变量的总和，其数值的大小将指定一个能够确保所有数值为正数的偏移量。

如果将采样不足的数据的重建过程建立在一个好的网格操作基础上，那么就可以大大提高重建效果。如果已知 PSF 具有较高的分辨率，那么 deconvlucy 函数将使用 SUBSMPL 参数来指定采样不足的比例。如果数据采样不足是由于图像获取过程中的镜头像素装仓问题产生的，那么每个像素观察到的 PSF 就是一个好的网格 PSF。另外，PSF 还可以通过观察自像素偏移或光学模型技术获得。这种方法对星(高信噪比)图像尤为有效，因为星可以被有效地限制在像素的中心位置。如果星位于两个像素之间，那么它将被作为邻域像素的组合进行重建。一个好的网格将会使星扩散流序列重新朝向图像的中心。

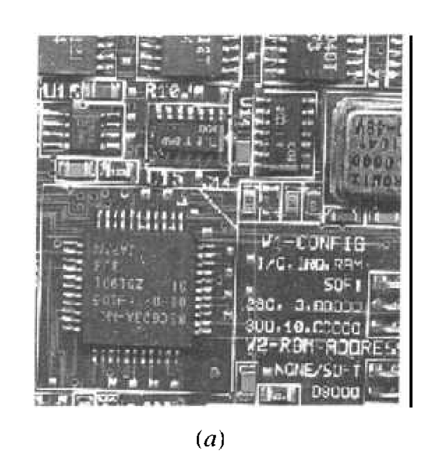
下面给出三个实例说明 deconvlucy 函数的具体使用方法。

**例 9.8**：调用 deconvlucy 函数对图 9.17(*b*)所示的有噪声模糊图像(原始图像如图 9.17(*a*)所示)进行重建，要求尽量获得较好的效果。

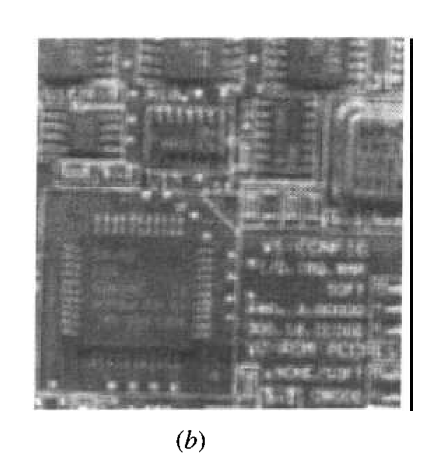
限制算法的重复次数为 5 次，使用以下代码进行图像复原：

```
PSF = fspecial('gaussian',5,5);
luc1 = deconvlucy(BlurredNoisy,PSF,5);          % BlurredNoisy 为待处理图像
subplot(2,2,1); imshow(luc1);
```

复原结果如图 9.18(*a*)所示。由于每一次重复过程得到的输出图像都是不同的，因此可以使用交互式的图像复原方法，也就是通过观察指定重复次数得到的复原图像，修改使用的 deconvlucy 函数参数再进行下一步的复原过程，最终得到较好的重建图像。例如，本例中将使用以下代码对以上生成的重建图像做进一步的复原操作：

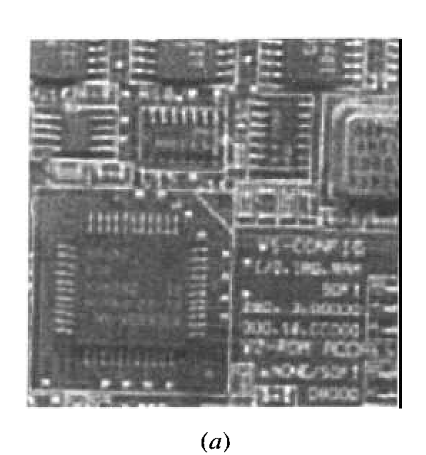


(a)

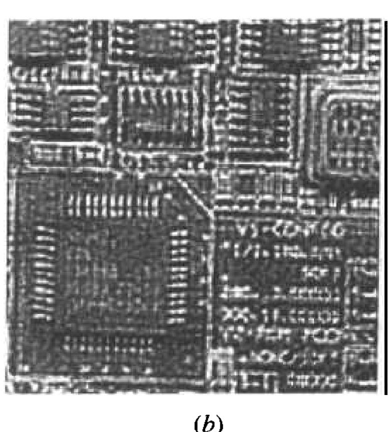


(b)

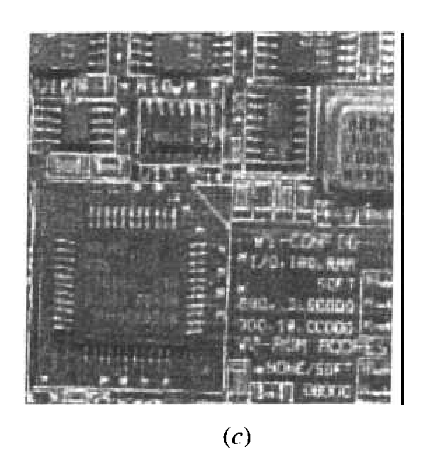
图 9.17 原始图像及其有噪声模糊化图像
(a) 原始图像；(b) 有噪声模糊图像



(a)



(b)



(c)

图 9.18 不同复原方法的显示效果比较
(a) 重复 5 次；(b) 重复 15 次；(c) 噪声控制

```
luc2_cell = deconvlucy(luc1_cell,PSF);
luc2 = im2uint8(luc2_cell{2});
subplot(2,2,2);imshow(luc2);
```

复原效果如图 9.18(b)所示。从图中可以看出，进一步重建后的结果反而出现了更多的噪声，这是由于重建过程过于逼近原始图像而导致原始图像的噪声放大而造成的。我们使用以下代码来控制噪声，得到的重建图像如图 9.18(c)所示：

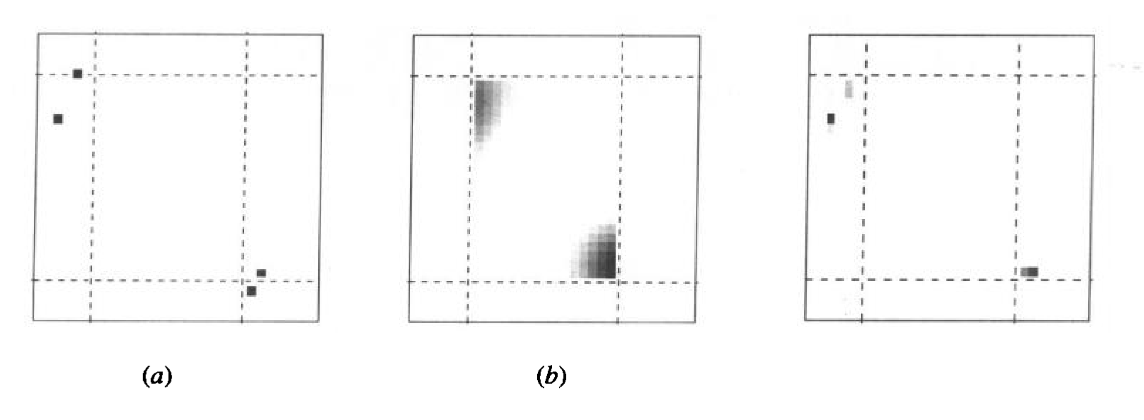
```
v=.002;
DAMPAR = im2uint8(3 * sqrt(V));
luc3 = deconvlucy(BlurredNoisy,PSF,15,DAMPAR);
subplot(2,2,3);imshow(luc3);
```

**例 9.9**：对图 9.19(b)所示的模糊星图像(原始图像如图 9.19(a)所示)进行重建。

对于图 9.19(b)所示的模糊星图像，以下代码仅使用邻域中的中心像素加权值进行计算，其他像素都从最优化过程中排除出去。

```
WT = zeros(32);
WT(6:27,8:23) = 1;
luc4 = deconvlucy(CutEdged,PSF,300,0,WT);
figure;imshow(1-luc4,[],'notruesize');
set(gca,'Visible','on','XTickLabel',[],'YTickLabel',[],...
    'XTick',[7 24],'XGrid','on','YTick',[5 28],'YGrid','on');
```

重建结果如图 9.20 所示。



(*a*)　　(*b*)

图 9.19　原始图像及其模糊星图像
(*a*) 原始图像；(*b*) 模糊星图像

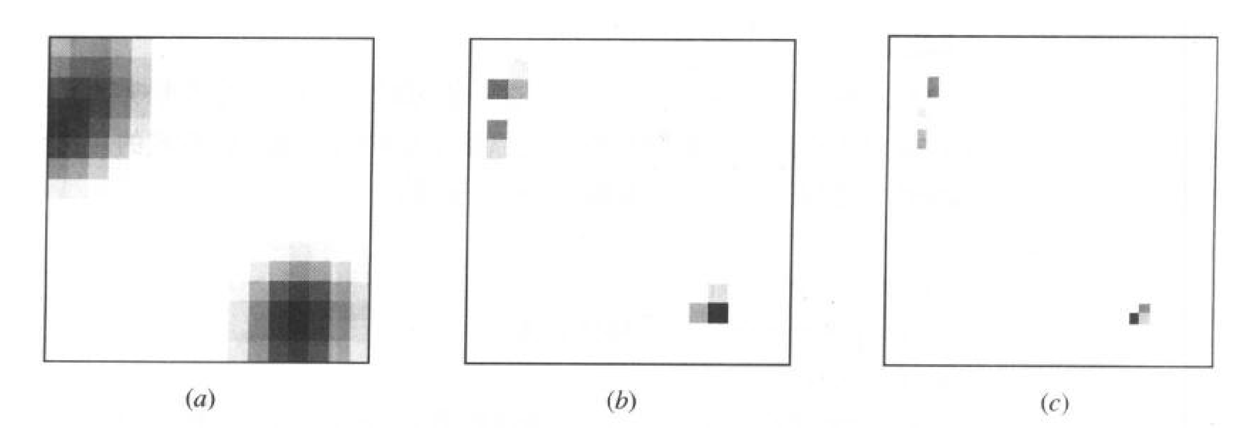
图 9.20　模糊星图像复原后的效果

**例 9.10**：对图 9.21(*a*)所示的采样不足图像(原始图像如图 9.19(*a*)所示)进行重建。

这里首先要介绍一下如何利用一个将导致数据采样不足的 PSF 把图 9.19(*a*)模拟成一幅采样不足图像的方法，其程序代码如下：

```
Binned = squeeze(sum(reshape(Blurred,[2 16 2 16])));  %Blurred 为模糊化的原始图像
BinnedImage = squeeze(sum(Binned,2));
Binned = squeeze(sum(reshape(PSF(1:14,1:14),[2 7 2 7])));
BinnedPSF = squeeze(sum(Binned,2));
subplot(221);imshow(1-BinnedImage,[],'notruesize');
set(gca,'Visible','on','XTick',[],'YTick',[]);
```

首先使用导致采样不足的 PSF 对图像进行复原，复原结果如图 9.21(*b*)所示。



(*a*)　　(*b*)　　(*c*)

图 9.21　重采样复原重建图像效果对比
(*a*) 采样不足图像；(*b*) 采样不足的 PSF 图像复原结果；(*c*) 真实 PSF 图像复原结果

```
luc5 = deconvlucy(BinnedImage,BinnedPSF,100);
subplot(222);imshow(1-luc5,[],'notruesize');
set(gca,'Visible','on','XTick',[],'YTick',[]);
```

显然，使用导致采样不足的 PSF 进行图像复原得到的结果是非常不满意的。以下代码将创建一个采样良好的 PSF(由原始图像到采样不足图像的真实 PSF)，并利用它对图像进行重建，其重建结果如图 9.21(*c*)所示。

```
PSF = fspecial('gaussian',15,3);
luc6 = deconvlucy(BinnedImage,PSF,100,[],[],[],2);
subplot(223);imshow(1-luc6,[],'notruesize');
set(gca,'Visible','on','XTick',[],'YTick',[]);
```

### 9.4.6 盲去卷积复原

MATLAB 的 deconvblind 函数使用类似于加速收敛 Lucy-Richardson 算法的执行过程来同时重建图像和 PSF。盲去卷积算法可以在对失真情况(包括噪声和模糊)毫无所知的情况下进行复原操作。deconvblind 函数与 deconvlucy 函数一样，可以实现多种基于原始 Lucy-Richardson 最大化可能性算法的复杂图像重建修改算法。deconvblind 函数的调用格式如下：

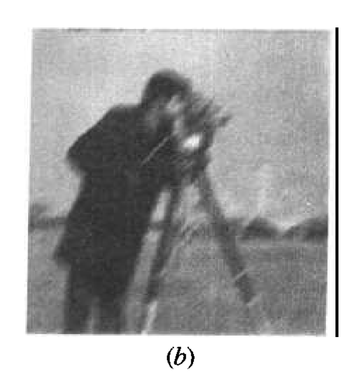
```
[J,PSF] = DECONVBLIND(I,INITPSF,NUMIT,DAMPAR,WEIGHT,READOUT)
```

其中，I 表示输入图像，INITPSF 表示 PSF 的估计值，NUMIT 表示算法重复次数，DAMPAR 表示偏移阈值，WEIGHT 用来屏蔽坏像素，READOUT 表示噪声矩阵。输出参数 J 表示复原后的图像，PSF 与 INITPSF 具有相同的大小，表示重建点扩散函数。下面通过一个实例说明该函数的使用方法。

**例 9.11**：对图 9.22(*b*)所示的图像(原始图像如图 9.22(*a*)所示)进行重建。



(*a*)



(*b*)

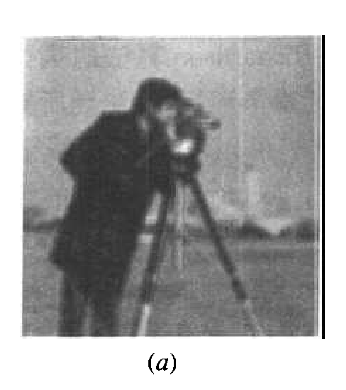
图 9.22 原始图像及其模糊噪声图像

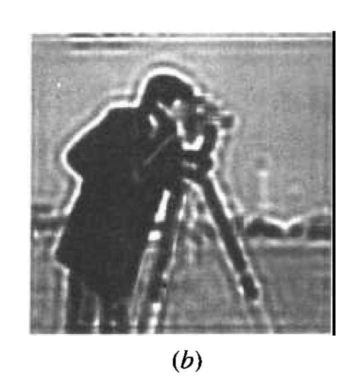
(*a*) 原始图像；(*b*) 模糊噪声图像

在调用 deconvblind 函数进行图像复原时，INITPSF 的大小是非常重要的一个指标。为了说明这一点，以下代码首先使用一个较小的 PSF(该 PSF 数组的每一维都比真实 PSF 少 4 个像素)对图像进行重建，然后再使用一个较大的 PSF(该 PSF 数组的每一维都比真实 PSF 多 4 个像素)进行复原，最后与采用真实大小的 PSF 进行比较：

```
PSF = fspecial('gaussian',7,10);
UNDERPSF = ones(size(PSF)-4);
[J1 P1] = deconvblind(Blurred,UNDERPSF);
subplot(1,3,1);imshow(J1);
OVERPSF = padarray(UNDERPSF,[4 4],'replicate','both');
[J2 P2] = deconvblind(Blurred,OVERPSF);
subplot(1,3,2);imshow(J2);
INITPSF = padarray(UNDERPSF,[2 2],'replicate','both');
[J3 P3] = deconvblind(Blurred,INITPSF);
subplot(1,3,3);imshow(J3);
```

复原后的图像分别如图 9.23(*a*)、(*b*)、(*c*)所示。


(*a*)


(*b*)


(*c*)

图 9.23　采用不同大小的 PSF 对图像进行复原的显示效果比较
(*a*) 较小的 PSF；(*b*) 较大的 PSF；(*c*) 真实的 PSF

从图 9.23 可以看出 PSF 对图像复原质量有着非常重要的影响。在实际应用中，可以通过分析，使用不同大小的 PSF 对图像进行重建，从中选择一个最合适的 PSF 值。例如，使用以下代码显示图 9.23(*a*)和图 9.23(*b*)返回的重建 PSF 以及真实的 PSF：

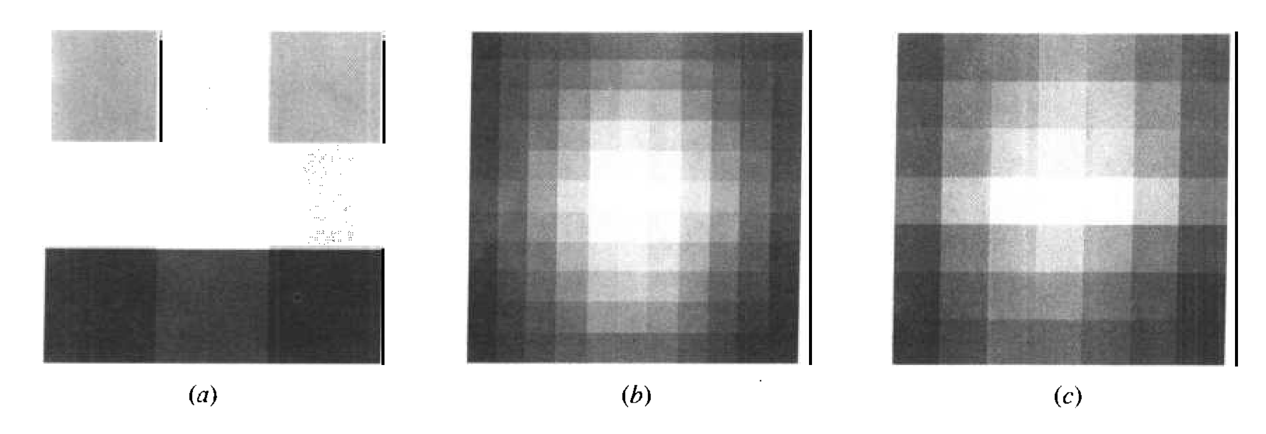
```
figure;
subplot(1,2,1);imshow(P1);
subplot(1,2,2);imshow(P2);
```

显示结果如图 9.24 所示。分析图 9.24(*a*)和图 9.24(*b*)可知，真实的 PSF 大小必定是介于这两者之间的，这样我们就可以选择一个合适的 PSF 值了。

图 9.23 所示的复原图像都存在一定的“环”，这些环是由图像灰度变换较大的部分或图像边界产生的。下面介绍如何使用 WEIGHT 参数来消除环的存在，以提高图像的复原质量。

首先要调用 edge 函数找出图像中灰度变化较大的部分。根据先验知识，选择灰度变化阈值为 0.3。同时对图像进行膨胀操作以扩充图像的处理区域。灰度变换较大的像素和图像的边界像素都将被设置为数值 0。

```
WEIGHT = edge(I,'sobel',.3);
se = strel('disk',2);
WEIGHT = 1-double(imdilate(WEIGHT,se));
WEIGHT([1:3 end-[0:2]],:) = 0;
```



(a) (b) (c)

图 9.24 不同大小 PSF 的图像比较
(a) 较小的 PSF；(b) 较大的 PSF；(c) 真实的 PSF

```
WEIGHT(:,[1:3 end-[0:2]]) = 0;
figure;imshow(WEIGHT);title('Weight array');
```

最终得到的 WEIGHT 矩阵如图 9.25 所示。



图 9.25 权矩阵

然后使用以上定义的 WEIGHT 数组对图像进行重建：

```
[J P] = deconvblind(Blurred,INITPSF,30,[],WEIGHT);
figure;subplot(1,2,1);imshow(J);
```

重建的效果如图 9.26(a)所示。此时的复原效果仍然不是很好，需要对 PSF 进行修改，从而进一步提高图像的复原质量。这里将从以上使用的 PSF 的每一维中减去 P1 和 P2 个像素，然后使用零填充方法将得到的 PSF 恢复为真实PSF 大小，得到的结果再用来进行图像重建，其程序代码如下：

```
str = 'padarray(PSF(P1+1:end-P1,P2+1:end-P2),[P1 P2])';
FUN = inline(str,PSF,'P1','P2');
[JF PF] = deconvblind(Blurred,OVERPSF,30,[],WEIGHT,FUN,2,2);
subplot(1,2,2);imshow(JF);
```

最终的复原图像如图 9.26(b)所示。



(*a*)



(*b*)

图 9.26　PSF 修改前、后的复原图像的显示效果比较
(*a*) 初步重建效果；(*b*) PSF 修正后的重建效果

## 【习　题】

1. 实现图 9.3 和 9.4 所示的魏纳滤波效果。

2. 通过比较 MATLAB 四种图像复原实现方法的效果，分别说明各种方法的适用范围。

# 第十章 图像编码与压缩

**本章要点：**

★ 图像编码概述

★ 无损压缩技术

★ 有损压缩技术

★ 图像压缩国际标准

## 10.1　图像编码概述

### 10.1.1　图像编码与压缩概念

各种媒体信息，特别是图像的数据量是非常大的。例如，一幅 640×480 分辨率的 24 位真彩色图像的数据量约占 900 KB 的存储空间，这样大的数据量不仅对计算机的存储和处理能力提出了要求，而且也使得图像通信的信道传输速率受到限制。因此，为了存储、处理和传输这些数据，必须要对图像信息进行压缩。

在保证一定图像质量的前提下，采取某种编码方式，以尽量减少图像的比特数，这便是图像压缩的研究课题。另外，在研究图像数据压缩编码的过程中，还要考虑的一个重要问题是重建后图像的质量问题。如果一幅图像的编码率很低，但是重建后效果却不能让人满意，那么这种编码就是毫无意义的。从视频图像开始出现到现在，图像品质及其度量方法一直都是人们研究的热点问题。图像品质的核心问题是逼真度，一般采用压缩处理后的图像与标准图像间的偏差作为图像逼真度的量度，偏差包括亮度、对比度、色度、分辨率等参数。

图像数据的压缩基于这样两个特点：一个特点是图像信息存在着很大的冗余度，数据之间存在着相关性，如相邻像素之间色彩的相关性等。一般来说，原始图像越有规律，像素间的相关性就越强，可以压缩的数据就越多；另一个特点是，由于人眼是图像信息的接收端，所以可利用视觉对于边缘急剧变化不敏感（视觉掩盖效应），以及对图像的亮度信息敏感、对颜色分辨率弱等特点来实现对图像的高压缩比，使得解压后的图像信号仍有着满意的质量。

由图像的这两个特点发展而来的数据压缩基本方法有两类：一类是将相同的或相似的数据或数据特征归类，使用较少的数据量描述原始数据，以达到减少数据量的目的，这种压缩一般被称为无损压缩；第二类方法是利用人眼的视觉特性有针对性地简化不重要的数据，以减少总的数据量，这种压缩一般被称为有损压缩，只要损失的数据不会影响人眼主

观接收的效果，就可采用这种压缩方法。

数据压缩技术也称为编码技术，它是一种很复杂的数学运算过程，从 1948 年 Oliver 提出 PCM 编码理论开始，至今已有 50 多年的研究历史。基于不同的信号源、不同的应用场合，已经提出了不同思路和技术的编码方法。实际图像的特性是多种多样的，各像素之间的关系也很复杂，为了寻找有效的编码方法，必须对原始图像的内部结构有比较清楚的了解。

### 10.1.2 图像信息熵和信息冗余度

表面看起来信息似乎是一种无法度量的抽象概念，但是如果从概率统计学的角度来看，信息出现的概率还是可以度量的，这个度量方法就是信息量。一般将信息量定义为信息源发出的所有消息中该信息出现概率的负对数，单位为：比特，即：

$$I(l_i) = -\operatorname{lb} p(l_i) \tag{10.1}$$

式(10.1)中之所以加一个负号是因为 $p(l_i)$ 总是小于等于 1 的，所以在式(10.1)中加一个负号以保证信息量为正。由于信息量 $I(l_i)$ 可以看成是离散随机变量，所以可以给信息量定义一些统计量。例如，平均信息量可以定义为 $I(l_i)$ 的数学期望：

$$H(l) = \sum_{i=0}^{m-1} p(l_i) I(l_i) = -\sum_{i=0}^{m-1} p(l_i) \operatorname{lb} p(l_i) \tag{10.2}$$

$H(l)$ 就是信息熵，其单位为比特/符号。例如，如果从 64 个数中选出某一个数，可先问“是否大于 32?”以消除半数的可能，这样只要 6 次就可选出该数。这是因为每提问一次都会得到 1 bit 的信息量。由此可见，在 64 个数中选定某一数所需的信息量是 lb64＝6 bit。

### 10.1.3 图像逼真度和质量

图像的压缩过程通常都是根据逼真度来设计编码方法的。逼真度是衡量图像品质的核心参数，一般采用压缩处理后的图像与标准图像间的偏差作为图像逼真度的量度。偏差包括亮度、对比度、色度、分辨率等一些参数。常用的逼真度准测有两种，一是客观逼真度准则，二是主观逼真度准则。

当损失的信息量可以用编码输入与解码输出图像的函数表示时，就称该编码是基于客观逼真度原则的。最常用的客观逼真度准则是输入、输出图像间的均方根误差准则。令输入图像和经过压缩及解压的图像分别为 $f(x,y)$ 和 $\hat{f}(x,y)$，假设两幅图像大小均为 $M \times N$，那么 $f(x,y)$ 和 $\hat{f}(x,y)$ 之间的均方误差定义为

$$e_{\mathrm{rms}} = \left[\frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}(\hat{f}(x,y) - f(x,y))^2\right]^{1/2} \tag{10.3}$$

输出图像的均方信噪比也是一种常用的客观保真度准则，其定义如下：

$$SNR_{\mathrm{rms}} = \frac{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\hat{f}(x,y)^2}{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[\hat{f}(x,y) - f(x,y)]^2} \tag{10.4}$$

式(10.4)的平方根称为均方根信噪比 $SNR_{\mathrm{rms}}$。在实际应用中常将 $SNR$ 进行归一化，其单位是分贝。令

$$\bar{f} = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)$$

则

$$SNR = 10\lg\left[\frac{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[\hat{f}(x,y)-\bar{f}]^2}{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[\hat{f}(x,y)-f(x,y)]^2}\right] \tag{10.5}$$

因为对图像进行压缩和解压的最终目的还是要提供给人眼进行观察的，所以使用主观方法度量图像的质量也是非常有效、可行的。常用的主观保真度度量方法是：向一组(通常要多于 20 位)观察者展示一幅图像，将观察者的评价综合起来进行统计分析，得到主观评价结果。评价可以按照某种绝对尺度(一幅图像的观感)进行，也可以通过比较两幅图像(标准图像与解压图像)进行，评价结果一般将图像分为优秀、良好、可用、勉强可看、差和不能使用几种。

### 10.1.4　图像编码的常用方法

数据压缩方法种类繁多，总体上可以分为无损压缩和有损压缩两大类。

无损压缩利用数据的统计冗余进行压缩，可完全恢复原始数据而不引入任何失真，但压缩率受到数据统计冗余度的理论限制，压缩比一般为 2∶1～5∶1，这类方法广泛用于文本数据、程序和特殊应用场合的图像数据(如指纹图像、医学图像等)的压缩。由于受到压缩比的限制，因而仅使用无损压缩方法是不可能解决图像和数字视频的存储和传输问题的。

有损压缩方法利用了人类视觉对图像中的某些频率成分不敏感的特性，允许压缩过程中损失一定的信息。虽然经有损压缩后的图像不能完全恢复成原始图像的状态，但是损失的部分对理解原始图像的影响较小，却换来了大得多的压缩比。有损压缩方式被广泛应用于语音、图像和视频等的数据压缩中。

数据压缩研究中应注意的问题是：首先编码方法必须能用计算机或 VLSI 硬件电路高速实现；其次要符合当前的国际标准。Shannon-Fano 编码、Huffman 编码、行程(Run-length)编码、LZW 编码(Lempel - Ziv - Welch)和算术编码等都是经常使用的无损压缩方法，而变换编码、预测编码则是比较常用的有损压缩方法。新一代的数据压缩方法，如基于模型的压缩方法、分形压缩和小波变换方法等也已经接近实用化水平。

## 10.2　无损压缩技术

### 10.2.1　无损压缩技术概述

无损压缩算法可以分为两个大类：一种是基于字典的编码方法，另一种是基于统计的编码方法。

基于字典的编码方法生成的压缩文件包含的是定长码，即采用相同的位数(bit)对数据进行编码。大多数存储数字信息的编码系统都采用定长码。例如，常用的 ASCII 码就是定长码，其码长为 1(单位为 Byte)。汉字国标码也是定长码，其码长为 2。基于字典编码方法

生成的每个码都代表原文件中数据的一个特定序列，常用的压缩方法有行程编码和 LZW 编码等。

基于统计方法生成的压缩文件包含的是变长码，即采用不相同的位数对数据进行编码，以节省存储空间。不同的字符或汉字出现的概率是不同的，有的字符出现的概率非常高，有的则非常低。据统计，英文字母中 E 的使用概率约为 13%，而字母 Z 的使用概率仅为 0.08%。另外，很多图像常含有单色的大面积图块，而且某些颜色比其它颜色出现更频繁。因此，为了节省空间，在对数据进行编码时，就有可能对那些经常出现的数据指定较少的位数来表示，而对那些不常出现的数据指定较多的位数来表示，这样从总的效果看还是节省了存储空间。例如，国际电报码中使用单点（“·”）来表示字母 e，而用横横点点（“——··”）表示字母 z。在实际应用中，最常用的统计编码方法是哈夫曼编码和算术编码方法等。

### 10.2.2 行程编码

某些图像，特别是计算机生成的图像往往包含许多颜色相同的块。在这些块中，许多连续的扫描行或者同一扫描行上有许多连续的像素都具有相同的颜色值。在这些情况下就不需要存储每一个像素的颜色值，而是仅仅存储一个像素值以及具有相同颜色的像素数目，将这种编码方法称为行程编码(Run-Length Encoding，RLE)，连续的具有相同颜色值的所有像素构成一个行程。像素的颜色值以及它的连续长度和终点位置是行程编码的重要参数。

RLE 编码技术相当直观和经济，运算也相当简单，因此解压缩速度很快。压缩率的大小取决于图像本身的特点。图像中具有相同颜色的横向色块越大、图像块数目越多，压缩比就越大，反之就越小。如果图像中有大量纵向色块，则可先把图像旋转 90°，再用 RLE 压缩，也可以得到较大的压缩比。RLE 压缩编码适用于计算机生成的图像，尤其是二进制图像，能够有效地减少存储容量。然而，由于自然图像往往是五光十色的，其行程长度非常短，若用 RLE 对它进行编码，不仅不能压缩数据，反而会造成更大的冗余，因此对复杂的图像都不能单纯地采用 RLE 进行编码。

### 10.2.3 哈夫曼(Huffman)编码

哈夫曼编码的基本方法是先对图像数据扫描一遍，计算出各种像素出现的概率，按概率的大小指定不同长度的惟一码字，由此得到一张该图像的哈夫曼码表。编码后的图像数据记录的是每个像素的码字，而码字与实际像素值的对应关系记录在码表中。当然，码表是附在图像文件中的。

哈夫曼编码过程如下：

(1) 将消息按照出现概率从大到小排列，记为：$p_1 \geqslant p_2 \geqslant \cdots \geqslant p_{m-1} \geqslant p_m$。

(2) 给最小的概率 $p_m$ 赋予符号 1，倒数第二小的概率 $p_{m-1}$ 赋予符号 0。

(3) 计算联合概率 $p_i = p_m + p_{m-1}$，将未处理的 $m-2$ 个概率与 $p_i$ 一起重新排序。

(4) 重复步骤(1)到步骤(3)，直到所有的概率都被赋予了一个符号为止。

下面举例说明哈夫曼编码的过程。表 10.1 给出了某个信号源发出的 8 个消息及其出现的概率。图 10.1 给出了哈夫曼编码的整个过程。

**表 10.1　消息及其出现的概率**

| 消息 | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 概率 | 0.1 | 0.18 | 0.4 | 0.05 | 0.06 | 0.1 | 0.07 | 0.04 |

从以上的编码过程可以看出，对于出现概率不同的消息，其码字长度也不同，概率大的信息码字短，概率小的码字长。这里要注意，消息对应的码字顺序是由其本身概率赋予的符号及其所构成的联合概率的符号的排列进行反序构成的。例如，消息 H 本身的概率符号为 1，联合概率 0.09 的符号为 1，0.19 为 0，0.37 为 0，0.60 为 0，则顺序为 11000，反序后得到的码字为 00011。

哈夫曼编码是一种最佳的编码方法，可以证明，采用这种编码方法得到的单元像素的比特数最接近图像的实际熵数。哈夫曼方法是一种较为实用的统计编码方法，其他更先进的统计方法虽然能够达到更高的压缩比，但是编码和解码时间都较长。

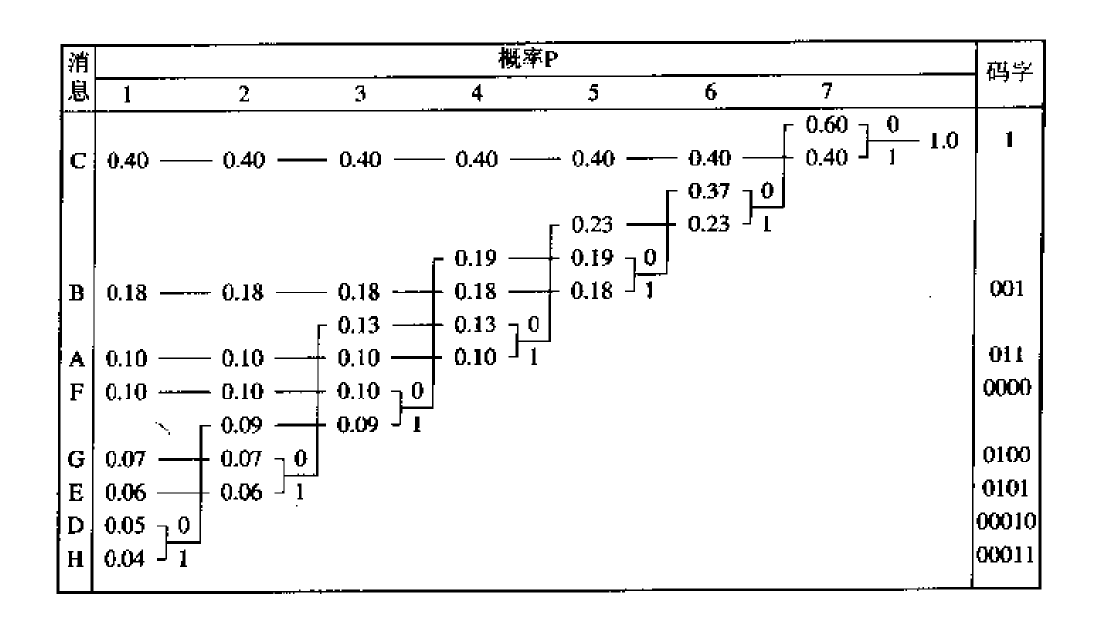



图 10.1　哈夫曼编码过程示意图

### 10.2.4　算术编码

算术编码是一种从整个消息序列出发，采用递推形式连续编码的方法。在算术编码方法中不存在消息与码字之间的对应关系，一个码字要赋给整个消息符号序列，码字本身将确定一个 0～1 之间的实数区间。下面给出一个算术编码过程的例子。

假设需要对字符串“ab bc”进行编码。在此字符串中，空格和字符 a、b、c 出现的概率分别为 0.2、0.2、0.4 和 0.2，我们定义其取值区间分别为[0.0 0.2)、[0.2 0.4)、[0.4 0.8)和[0.8 1.0)。在编码开始时，假设该字符串占据了整个区间[0.0 1.0)，那么第一个字符 a 对应的取值区间就是[0.2 0.4)，将这个取值区间扩充为整个字符串的取值区间；第二个字符 b 占据的取值空间就是[0.2 0.4)范围内的[0.4 0.8)部分，即[0.28 0.36)（说明：将[0.2 0.4)看作一个单位，该单位的[0.4 0.8)部分即为[0.28 0.36)，即 0.28＝0.2＋(0.4－0.2)×0.4；0.36＝0.2＋(0.4－0.2)×0.8；【下同】；第三个字符——空格所

占据的空间就是[0.28 0.36)的[0.0 0.2)部分，即[0.28 0.296)；第四个字符 b 的取值空间为[0.2864 0.2928)；最后一个字符 c 的取值空间为[0.29152 0.2864)。最后得到的取值空间[0.29152 0.2864)中的任何一个实数都可以代表这个字符串，这样就将一个字符串用一个实数来表示了，使数据量大大减小。

假设取 0.29152 作为码字，那么以上算术编码对应的解码步骤如下：

(1) 根据码字所在范围确定当前码字的第一个字符并输出：0.29152 位于范围[0.2 0.4)之间，所以输出第一个字符 a；

(2) 消除已译字符对码字的影响。此时不能再使用 0.29152 作为码字，必须将字符 a 对该码字的影响从 0.29152 中消除。具体做法是这样的：将 a 的取值下限 0.2 从码字 0.29152 中减掉，再除以 a 的取值宽度 0.2，得到新的码字 0.4576；

(3) 重复步骤(1)和(2)的做法，直到码字处理完成。

从以上的介绍可以看出，在算术编码过程中仅用到了代数运算和移位运算，所以称这种编码方法为算术编码。当需要编码的字符序列长度增加时，运用算术编码将会接近无损编码的理论极限。算术编码之所以不能完全达到极限的原因有二：一是编码需要添加最终序列的结束标志；另一个原因是运算过程中的操作精度有限(由除法引起)。

### 10.2.5　无损编码技术的 MATLAB 实现方法

MATLAB 的图像处理工具箱并没有提供直接进行图像编码的函数或命令，这是因为 MATLAB 的图像输入、输出和读、写函数能够识别各种压缩图像格式文件，利用这些函数就可以间接地实现图像压缩。但是为了说明图像的编码过程，我们利用 MATLAB 的基本语法和某些基本图像函数来进行编码实现。

首先来看行程编码。在以上介绍的三种编码方法中，行程编码是最简单、最容易实现的。进行行程编码的方法可以是多种多样的，以下代码将每一个不同行程(即不同颜色的像素块)的起始坐标和灰度值都记录下来：

```
I=imread('code.gif');
[m n]=size(I);
c=I(1,1);E(1,1)=1;E(1,2)=1;E(1,3)=c;
t1=2;
for k=1:m
    for j=1:n
        if(not(and(k==1,j==1)))
            if(not(I(k,j)==c))
                E(t1,1)=k;E(t1,2)=j;E(t1,3)=I(k,j);
                    c=I(k,j);
                    t1=t1-1;
            end
        end
    end
end
```

编码后的图像存储在变量 E 中，该变量是一个三维数组，前两维表示起始像素的横、纵坐标，第三维表示该行程的颜色值。通过调用 imfinfo 函数观察返回变量 info 可以知道，原始图像的大小为 175×123，在 MATLAB 中的位深度为 8，所以存储该图像文件需要 21 525 个字节，而调用 whos 命令可以知道 E 是一个 205×3 的数组，所以 E 仅仅占用 615 个字节，这就大大减少了图像的存储空间。

下面介绍哈夫曼编码的实现方法。进行哈夫曼编码首先要统计图像中各种颜色值出现的概率，然后再进行排序编码。这种编码方法较为复杂，但是相对于行程编码方法而言，其效果要好得多。哈夫曼编码的 MATLAB 实现代码如下：

```
[m,n]=size(I);
p1=1; s=m*n;
for k=1:m                    %获取图像中的颜色总数
    for l=1:n
        f=0;
        for b=1:p1-1
            if(c(b,1)==I(k,l)) f=1; break; end
        end
        if(f==0) c(p1,1)=I(k,l); p1=p1+1; end
    end
end
for g=1:p1-1                 %计算各种颜色值出现的概率
    p(g)=0;c(g,2)=0;
    for k=1:m
        for l=1:n
            if(c(g,1)==I(k,l)) p(g)=p(g)+1; end
        end
    end
    p(g)=p(g)/s;
end
pn=0; po=1;
while(1)                     %按照概率排序生成一个符号(0 或 1)树并记录各结点
    if(pn>=1.0) break;
    else
        [pm p2]=min(p(1:p1-1)); p(p2)=1.1;
        [pm2,p3]=min(p(1:p1-1)); p(p3)=1.1;
        pn=pm+pm2; p(p1)=pn;
        tree(po,1)=p2; tree(po,2)=p3;
        po=po+1; p1=p1+1;
    end
end
for k=1:po-1                 %沿符号树进行搜索生成哈夫曼编码
    tt=k; m1=1;
    if(or(tree(k,1)<9,tree(k,2)<9))
```

```
            if(tree(k,1)<9)
                c(tree(k,1),2)=c(tree(k,1),2)+m1;
                m2=1;
                while(tt<po-1)
                    m1=m1*2;
                    for l=tt:po-1
                        if(tree(l,1)==tt+g)
                            c(tree(k,1),2)=c(tree(k,1),2)+m1;
                            m2=m2+1; tt=l; break;
                        elseif(tree(l,2)==tt+g)
                            m2=m2+1; tt=l; break;
                        end
                    end
                end
            c(tree(k,1),3)=m2;
            end
            tt=k; m1=1;
            if(tree(k,2)<9)
                m2=1;
                while(tt<po-1)
                    m1=m1*2;
                    for l=tt:po-1
                        if(tree(l,1)==tt+g)
                            c(tree(k,2),2)=c(tree(k,2),2)+m1;
                            m2=m2+1; tt=l; break;
                        elseif(tree(l,2)==tt+g)
                            m2=m2+1; tt=l; break;
                        end
                    end
                end
                c(tree(k,2),3)=m2;
            end
        end
    end
```

以上代码中的输出数组 c 的第一维表示颜色值，第二维表示代码的数值大小，第三维表示该代码的位数，将这三个参数作为码表写在压缩文件头部，则其以下的数据将按照这三个参数记录图像中的所有像素颜色值，于是就可以得到哈夫曼编码的压缩文件。这里要注意的是，由于 MATLAB 不支持对某一位(bit)的读和写，所以利用该码表生成的每一个码字实际上还是 8 位的，最好使用其它软件(例如，C 语言等)进行改写，以实现真正的压缩。事实上 MATLAB 将图像写成 JPEG 文件也是用 C 语言实现的。

有关算术编码的实现方法留给读者思考。

# 10.3 有损压缩技术

## 10.3.1 预测编码

数字化的图像按照行或列重新排列后就可以得到一个一维信号序列。预测编码就是根据这个信号序列的一些已知情况来预测以后信号可能发生的情况，然后对预测的误差，而不是直接对信号进行编码。当预测比较准确、误差较小时，这种编码方式就能够达到数据压缩的目的。

下面首先讨论一下信号预测估计的一般方法。假设，已知 $x_n$ 以前的信号 $x_{n-1}$，$x_{n-2}$，…，$x_{n-m}$的值，那么可以得到 $x_n$ 的预测估计值为

$$\hat{x}_n = f(x_{n-1}, \cdots, x_{n-m}) \tag{10.6}$$

如果估计值 $\hat{x}_n$ 是信号 $x_{n-1}$，$x_{n-2}$，…，$x_{n-m}$的线性组合，那么就称该预测为线性预测，否则为非线性预测。常用的差分脉冲编码就是一种线性预测编码。

定义预测的误差为

$$e_n = x_n - \hat{x}_n \tag{10.7}$$

显然，如果函数 $f$ 选择的比较准确，那么 $x_n$ 和 $\hat{x}_n$ 的差值将比较小，那么传输这个差值将会比传输信号本身节省空间，这就是预测编码的基本原理。下面的问题就是怎样根据一定的最优标准来定义预测估计值函数。一般都采用均方误差最小作为预测的最优准则，预测误差的统计均方值就是 $e_n^2$ 的期望：

$$E(e_n^2) = \int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty}(x_n - \hat{x}_n)^2 p(x_n x_{n-1}\cdots x_{n-m})\,\mathrm{d}x_n\cdots\mathrm{d}x_{n-m} \tag{10.8}$$

对于线性预测来说，问题就转化为怎样利用均方误差最小准则求取预测系数 $a_i$。令均方误差对每个预测系数的偏导都为零：

$$\frac{\partial E(e_n^2)}{\partial a_i} = \frac{\partial}{\partial a_i}E\{[x_n - (a_1x_1 - a_2x_2 + \cdots + a_{n-1}x_{n-1})]^2\} = 0 \tag{10.9}$$

于是可以得到 $N-1$ 个方程组：

$$E\{[x_n - (a_1x_1 + a_2x_2 + \cdots + a_{n-1}x_{n-1})]x_i\} = 0 \tag{10.10}$$

解这个方程组就可以求出 $N-1$ 个预测系数。事实上虽然利用这个方程组可以计算出一个较为精确的线性预测函数，但是这个函数仅仅是针对这一幅图像而言的，适用范围较小而计算又较为复杂，所以通常并不使用这种解方程组的方法。对于一幅二维图像，常常使用以下简化预测公式进行预测

$$f(m,n) = \frac{1}{2}f(m,n-1) + \frac{1}{4}f(m-1,n) + \frac{1}{8}f(m-1,n-1) + \frac{1}{8}f(m-1,n+1) \tag{10.11}$$

式(10.11)中的系数总和为 1，这是为了保持图像的平均亮度不变。

在上述的线性预测编码中，如果使用同一行中的前面几个信号进行估计，那么称这种估计方法为一维预测方法，如果使用几行内的信号来估计，那么就称该方法为二维预测方法。一般都要利用二维预测方法通过消除行与行之间的相关性来进一步地压缩信号。二维

预测系数也是通过最小均方差准则计算得出的。当然，如果利用图像帧之间的相关性来进行预测编码，那么就可以获得更大的压缩比，这种压缩方法的效率与图像的运动变化规律密切相关。非线性预测主要是指预测系数变化的情况，这种预测编码方法能够更好地匹配图像的局部特征，其压缩效果通常要好一些。

## 10.3.2 变换编码

变换编码就是对图像数据进行某种形式的正交变换，并对变换后的简单数据进行编码，从而达到数据压缩的目的。无论是对单色图像、彩色图像、静止图像，还是运动图像，变换编码都能够获得较好的压缩比。

变换编码的基本过程是将原始图像分块，然后对每一块进行某种形式的正交变换，也可以简单地理解为将小块图像由时域变换到频域，使变换图像的能量主要集中在直流分量和低频分量上。在误差允许的条件下，用直流和部分低频分量来代表原始数据，从而达到数据压缩的目的。在解压缩时，利用已压缩的数据计算并补充高频分量，经过逆变换就可恢复原始数据。显然，变换编码减少了图像的信息熵，造成了信息量的减少，从而带来了一定的图像失真。

变换编码采用的正交变换种类很多，比如傅立叶变换、沃尔什哈达玛变换、哈尔变换、斜变换、余弦变换、正弦变换，还有基于统计特性的 K-L 变换等。K-L 变换后的各系数相关性小，能量集中，压缩生成的误差最小，但是计算复杂，执行速度慢。由于离散余弦变换(DCT 变换)与 K-L 变换性能最为接近，而且具有易于硬件实现的快速算法，所以得到了广泛的应用。在前面的章节中已经介绍过 DCT 变换的概念，下面主要介绍 DCT 在变换编码中的具体应用。

用 DCT 变换实现图像的压缩编码需经过变换、压缩和编码三个步骤。二维 DCT 变换编码压缩和解压缩的框图如图 10.2 所示。

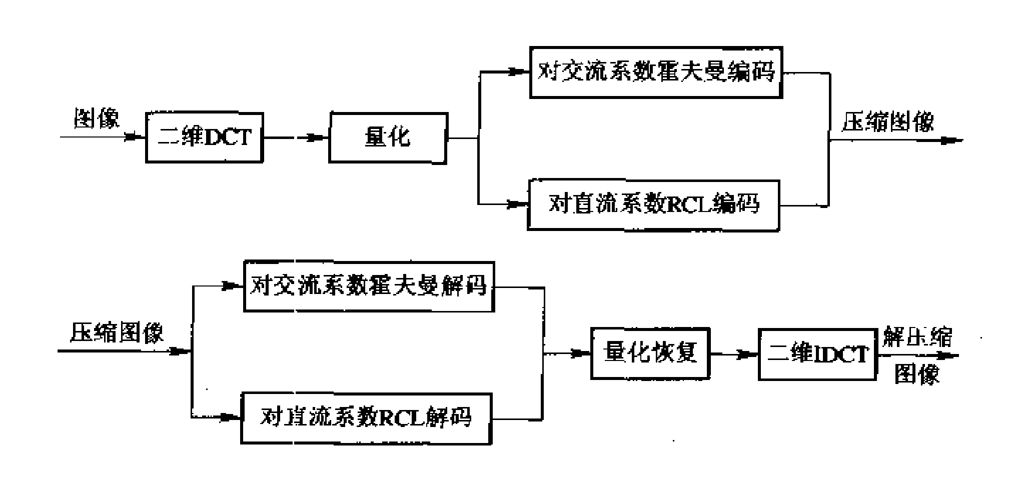



图 10.2　二维 DCT 变换编码压缩和解压缩框图

利用二维 DCT 进行图像数据压缩时，首先要将输入图像分成若干 $N \times N$ 的图像块。由于 $N$ 取值小到一定程度时，采用变换处理可能会出现块与块之间边界上存在边界效应的现象(即存在不连续点)。当 $N<8$ 时，边界效应比较明显，所以要求 $N \geq 8$。在实际应用中一般取 $N=8$。二维快速 *DCT* 则是把 $8 \times 8$ 个图像块不断分成更小的无交叠子块，然后

直接再对数据块进行运算操作。

8×8 数据块将输入分解成 64 个正交基信号后，每个基信号都对应于 64 个独立二维空间中的某一个频率。DCT 变换编码输出这 64 个 DCT 变换系数值(即基信号的幅值)，这些变换系数中包括一个代表直流分量的 DC 系数和 63 个代表交流分量的 AC 系数，DCT 变换的解压过程就是对这 64 个 DCT 变换系数进行逆变换运算，重建一个 64 点的输出图像。

为了达到压缩数据的目的，对 DCT 系数 $F(u,v)$ 还需作量化处理。量化就是通过减少精确度来减少存储整数所需比特数的过程。图像经过 DCT 变换压缩后，离原点(0，0)越远的元素对图像的贡献就越小，因而也就越不关心此处取值的精确性。量化首先要对每个系数确定一个量化步长(量化间隔)，然后用对应的量化步长去除对应的 DCT 系数并对其求整：

$$Fg(u,v)=\text{round}[F(u,v)/M(u,v)]$$

其中，$M(u,v)$ 是与 $F(u,v)$ 对应的每个 DCT 系数的量化步长，也称量化矩阵或量化表，是根据人类视觉系统和压缩图像类型的特点进行优化的系数矩阵。表 10.2 和 10.3 分别是 JPEG 标准中采用的亮度量化表和色度量化表。从量化表中可以看出，各变换系数的量化间隔是不一样的。对低频分量，量化间隔小，量化误差也小，因而精度较高；而频率愈高，量化间隔愈大，精度也就越低。这是因为高频分量只会影响图像的细节，从整体上说，没有低频分量重要。量化处理是一个多到一的映射，它是造成 DCT 编/解码信息损失的根源。

**表 10.2　JPEG 标准中的亮度量化表**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 16 | 11 | 10 | 16 | 24 | 40 | 51 | 61 |
| 12 | 12 | 14 | 19 | 26 | 58 | 60 | 55 |
| 14 | 13 | 16 | 24 | 40 | 57 | 69 | 56 |
| 14 | 17 | 22 | 29 | 51 | 87 | 80 | 62 |
| 18 | 22 | 37 | 56 | 68 | 109 | 103 | 77 |
| 24 | 35 | 55 | 64 | 81 | 104 | 113 | 92 |
| 49 | 64 | 78 | 87 | 103 | 121 | 120 | 101 |
| 72 | 92 | 95 | 98 | 112 | 100 | 103 | 99 |

**表 10.3　JPEG 标准中的色度量化表**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 17 | 18 | 24 | 47 | 99 | 99 | 99 | 99 |
| 18 | 21 | 26 | 66 | 99 | 99 | 99 | 99 |
| 24 | 26 | 56 | 99 | 99 | 99 | 99 | 99 |
| 47 | 66 | 99 | 99 | 99 | 99 | 99 | 99 |
| 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 |
| 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 |
| 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 |
| 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 |

对量化后的 DCT 系数按照概率进行曲徊排序，曲徊排序的路线如图 10.3 所示，然后将排序结果进行统计编码，一般都采用哈夫曼编码方法。

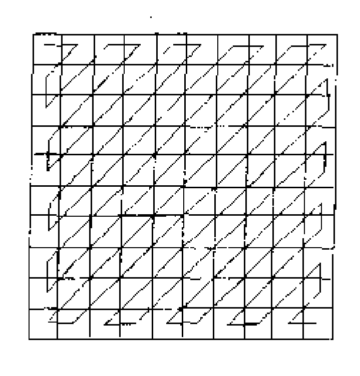


图 10.3 DCT 编码的曲徊排序路线

### 10.3.3 自适应编码

自适应编码是一种针对图像的某些局部或瞬间统计特性进行参数自动调整的编码方法。自适应编码并不是一种独立的编码方法，它总是结合其它某种编码方法进行具体应用的。例如，预测编码中的自适应预测编码，变换编码中的自适应变换编码等。

在预测编码中，当预测参数确定后，整个编码系统是固定不变的。但是当图像的内容发生变化时，如果能够将这些变化反馈给系统进行参数调节，那么必定会提高图像的质量和压缩比，这就是自适应预测编码的基本出发点。一般进行自适应预测编码都是在线性预测编码的基础上给预测函数添加一个自适应系数，在压缩过程中根据图像的变化特征修改这个自适应系数。例如，有人曾经提出过这样一种自适应预测编码技术：假设预测像素 $x$ 时的邻近像素排列如下：

$$x_6 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4 \rightarrow x_7 \rightarrow x_5 \rightarrow x_1 \rightarrow x$$

使用以下函数对 $x$ 进行预测

$$\hat{x} = k(a_1 x_1 + a_4 x_4)$$

其中，$a_1$、$a_4$ 为预测系数。$k$ 为自适应系数，按照以下定义取值：

$$k = \begin{cases} 1.125 & x_{n-1} = x_{\max} \\ 1.0 & x_{\min} < x_{n-1} < x_{\max} \\ 0.875 & x_{n-1} = x_{\min} \end{cases}$$

其中，$x_{n-1}$是第 $N-1$ 次抽样值的量化输出结果，$x_{\min}$是最小的量化输出值，$x_{\max}$是最大的量化输出值。从 $k$ 的取值可以看出，当 $x_{n-1}$位于最大与最小量化输出值之间时，就采用原有的预测函数；如果 $x_{n-1}$取最小的量化输出值时，那么 $k$ 将会使得 $x$ 的预测值提高 12.5%，这对于防止误差或斜率过载是有益的；而当 $x_{n-1}$取最大的量化输出值时，预测值将减小 12.5%，这对于提高图像平坦区域的质量是有益的。

自适应变换编码的出发点在于弥补单一变换编码存在的不足之处。对于 DCT 自适应变换编码而言，主要是添加了一种阈值编码，处理那些超过阈值的 DCT 系数不再是简单的抛弃，而是进行编码保存。另外，自适应技术还体现在图像块分割的大小上，图像块越小，所需的变换次数越多，效果越好，但是计算时间会增加。自适应 DCT 变换编码通过全面地衡量计算量、存储空间和编码效果，根据不同图像的不同特点来自动调节图像块的大小，以适应编码的要求。

### 10.3.4 有损压缩的 MATLAB 实现方法

以下代码将使用简化预测公式(10.11)进行线性预测编码。这里以灰度图像为例，通过使用 MATLAB 的文件读写函数 fopen、fwrite 和 fclose，将计算所得的误差以最小的位深度(在 MATLAB 中为 8 位)写入文件中。对于真彩色图像，只需对三个颜色通道调用以下

代码即可：

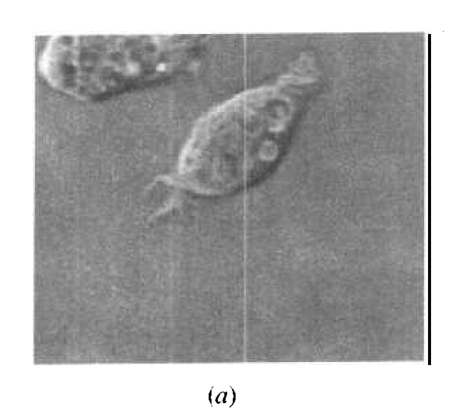
```
I=imread('image.ext');          %读入图像
fid=fopen('mydata.dat','w');
[m n]=size(I);
J=ones(m,n);
J(1:m,1)=I(1:m,1);
J(1,1:n)=I(1,1:n);
J(1:m,n)=I(1:m,n);
J(m,1:n)=J(m,1:n);
for k=2:m-1
    for l=2:n-1
        J(k,l)=I(k,l)-(I(k,l-1)/2+I(k-1,l)/4+I(k-1,l-1)/8+I(k-1,l+1)/8);
    end
end
J=round(J);
cont=fwrite(fid,J,'int8');
cc=fclose(fid);
```

显然，以上代码实现的压缩比为4:1(即双精度数据位数与8位符号整数位数的比值)。调用以下代码对以上预测编码文件进行解码，并通过显示原始文件和解压后的文件来比较压缩效果：

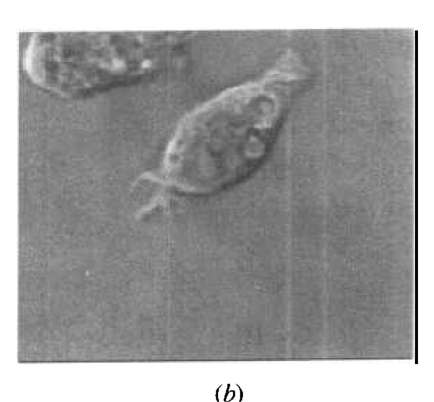
```
fid=fopen('mydata.dat','r');
I1=fread(fid,cont,'int8');
tt=1;
for l=1:n
    for k=1:m
        I(k,l)=I1(tt);
        tt=tt+1;
    end
end
I=double(I);
J=ones(m,n);
J(1:m,1)=I(1:m,1);
J(1,1:n)=I(1,1:n);
J(1:m,n)=I(1:m,n);
J(m,1:n)=I(m,1:n);
    for k=2:m-1
    for l=2:n-1
        J(k,l)=I(k,l)+(J(k,l-1)/2+J(k-1,l)/4+J(k-1,l-1)/8+J(k-1,l+1)/8);
    end
end
cc=fclose(fid);
J=uint8(J);
```

```
subplot(1,2,1),imshow(I2);
subplot(1,2,2),imshow(J);
```

原始图像如图 10.4(*a*)所示．编码后的解码图像如图 10.4(*b*)所示，两幅图像稍有差别。



(*a*)



(*b*)

图 10.4　图像预测编码前、后显示效果比较

(*a*) 编码前；(*b*) 编码后

以下代码将以真彩图像为例进行图像的变换编码：

```
I=imread('sea.bmp');
yiq=rgb2ntsc(I);
my=[16  11  10  16  24  40  51  61;12  12  14  19  26  58  60  55;
    14  13  16  24  40  57  69  56;14  17  22  29  51  87  80  62;
    18  22  37  56  68  109 103 77;24  35  55  64  81  104 113 92;
    49  64  78  87  103 121 120 101;72  92  95  98  112 100 103 99];
miq=[17  18  24  47  99  99  99  99;18  21  26  66  99  99  99  99;
     24  26  56  99  99  99  99  99;47  66  99  99  99  99  99  99;
     99  99  99  99  99  99  99  99;99  99  99  99  99  99  99  99;
     99  99  99  99  99  99  99  99;99  99  99  99  99  99  99  99];
I1=yiq(:,:,1); I2=yiq(:,:,2);
[m n]=size(I1);
t1=8; ti1=1;
while(t1<m)
    t1=t1+8; ti1=ti1+1;
end
t2=8; ti2=1;
while(t2<n)
    t2=t2+8; ti2=ti2+1;
end
times=0;
for k=0:ti1-2
    for l=0:ti2-2
        dct8x8(I1(k*8+1:k*8+8,l*8+1:l*8+8),my,times*64+1);
        dct8x8(I2(k*8+1:k*8+8,l*8+1:l*8+8), miq,times*64+1);
```

```
            times=times+1;
        end
        blkproc(I2(k*8+1:k*8+8,l*8+1:t2),[8 8],T);
    end
    for l=0:ti-2
    dct8x8(I1(k*8+1:t1,l*8+1:l*8+8),times*64+1);
        times=times+1;
    end
    dct8x8(I1(k*8+1:t1,l*8+1:t2),times*64+1);

    function dct8x8(I,m,s)
        T = inline('dctmtx(8)');
        y=blkproc(I,[8 8],T);
        y=round(y./m);
        p=1; te=1;
        while(p<=64)
            for q=1:te
                y1(s+p)=y(te-q+1,q); p=p+1;
            end
            for q=te:-1:1
                y1(s+p)=y(te-q+1,q); p=p+1;
            end
        end
        f=haffman(y1);
        c(s:s+64,1)=f(:,1) ; c(s:s+64,2)=f(:,2) ; c(s:s+64,3)=f(:,3)

    function c=haffman(I)
    ......
```

## 10.4　图像压缩的国际标准

### 10.4.1　二进制图像压缩标准

图像编码技术的发展和广泛应用促进了许多有关国际标准的制定，这些国标的制定工作主要是由国际标准化组织(ISO)和国际电信联盟(ITU，其前身为国际电报咨询委员会CCITT)进行的，这些组织根据需要处理的图像类型将图像压缩标准分为几个系列，其中包括二值图像压缩标准、静止灰度和彩色图像压缩标准、连续灰度和彩色图像压缩标准。

二值图像压缩标准主要有G3/G4和JBIG两种。G3和G4这两个二进制图像压缩标准最初是由CCITT的两个小组为图像传真应用而设计的，现在也应用于其他方面。G3标准主要采用一维行程编码技术，也间或使用二维行程编码技术；G4仅采用二维行程编码技术，是G3的一个简化版本。通过实验验证，G3对文字或少量绘图图像的压缩比约为

15：1，G4 比 G3 的压缩比要高将近一倍；JBIG 是由 G3、G4 两个设计小组共同制定的压缩标准，主要是针对 G3 和 G4 压缩中出现的半调灰度图像的扩展（不但不压缩，反而扩充了容量）问题而制定的。JBIG 采用自适应技术解决图像扩展问题，因而提高了 G3 和 G4 标准的编码效率，尤其是对于半调灰度图像，压缩比能够提高 2～30 倍。

### 10.4.2 静止图像压缩标准

专家组 JPEG（Joint Photographic Experts Group）经过五年艰苦、细致地工作后，于 1991 年 3 月提出了多灰度静止图像的数字压缩编码，即通常所说的 JPEG 标准，这是一种适用于彩色和单色多灰度或连续色调静止数字图像的压缩标准，包括 DPCM（差分脉冲编码调制）和有损压缩算法（DCT 和 Huffman 编码）两个部分，前者不会产生失真，但压缩比很小；后者虽然图像信息有所损失，但压缩比可以很大，压缩 20 倍左右时，人眼基本上看不出失真。

JPEG 标准实际上包含三个范畴：基本顺序过程主要是实现有损图像压缩，重建图像质量达到人眼难以观察出来的效果。采用的是 8×8 像素自适应 DCT 算法以及 Huffman 型的熵编码器；基于 DCT 的扩展过程使用累进工作方式，采用自适应算术编码过程；无失真过程采用预测编码及 Huffman 编码（或算术编码），可保证重建后的图像数据与原始图像数据完全相同。其中，基本顺序过程是 JPEG 最基本的压缩过程，符合 JPEG 标准的硬、软件编码/解码器都必须支持和实现这个过程，另两个过程是可选扩展，对某些特定的应用项目有很大的实用价值。

基本 JPEG 算法的操作可分成三个步骤：首先通过 DCT 去除数据冗余，然后使用量化表对 DCT 系数进行量化，最后对量化后的 DCT 系数进行编码使其熵达到最小，熵编码采用 Huffman 可变字长编码。

JPEG 采用 8×8 子块的二维离散余弦变换算法，在编码器输入端把原始图像（对彩色图像则是每个颜色分量）顺序地分割成一系列 8×8 子块。在图像块中，像素值一般变化较平缓，因此具有较低的空间频率。利用三维 8×8 DCT 可以将图像块的能量集中在极少数的几个系数上。

JPEG 标准中采用线性均匀量化器，量化过程为对 64 个 DCT 系数除以量化步长并进行四舍五入取整，量化步长由量化表决定。量化表元素因 DCT 系数位置和彩色分量的不同而取不同值。量化表为 8×8 矩阵，与 DCT 变换系数一一对应。量化表一般由用户规定（JPEG 标准中给出了参考值），并作为编码器的一个参数输入。量化表中的元素为 1～255 之间的任意整数，其值规定了其所对应的 DCT 系数的量化步长。DCT 变换系数除以量化表中对应位置的量化步长并舍弃小数部分后。

64 个变换系数经量化后，左上角的系数是直流分量（DC 系数），即空间域中 64 个图像采样值的均值。相邻 8×8 块之间的 DC 系数一般有很强的相关性，JPEG 标准对 DC 系数采用 DPCM 编码方法，即对相邻像素块之间的 L 系数的差值进行编码。其余 63 个交流分量（AC 系数）使用行程编码，从左上角开始沿对角线方向进行量化后的 AC 系数通常会有许多零值，以 Z 字形路径进行曲徊排序的行程编码方式有效地增加了连续出现的零值的个数。为了进一步压缩数据，对 DC 码和 AC 行程编码的码字可以再作基于统计特性的编码。JPEG 标准建议使用的统计编码方法有 Huffman 编码和自适应二进制算术编码两种。

目前JPEG专家组正在研究JPEG2000这一新的国际标准——采用小波变换的编码技术。JPEG2000标准较JPEG标准而言添加了诸如提高压缩质量、码流随机存储、结构开放、向下兼容等功能。

### 10.4.3　运动图像压缩标准

运动图像的国际压缩标准主要有H.261、MPEG－Ⅰ、MPEG－Ⅱ和MPEG－Ⅳ这几种，在此我们主要介绍MPEG系列标准。MPEG(Moving Pictures Experts Group)是ISO的一个小组，其工作兼顾了JPEG标准和CCITT专家组的H.261标准，于1990年形成了一个标准草案。MPEG算法除了对单幅图像进行编码外，还利用图像序列的相关特性去除帧间图像冗余，大大提高了视频图像的压缩比，在保持较高的图像视觉效果的前提下，使压缩比可以达到60～100倍左右。

MPEG压缩算法复杂、计算量大，其实现一般要有专门的硬件支持。MPEG标准分成两个阶段：第一个阶段(MPEG－Ⅰ)的目的是把221 Mbit/s的NTSC图像压缩到1.2 Mb/s，压缩率为200：1，这是图像压缩的工业认可标准；第二个阶段(MPEG－Ⅱ)的目标则是对每秒30帧的720×572分辨率的视频信号进行压缩，主要用于宽带传输的图像，压缩图像质量达到电视广播甚至HDTV的标准。和MPEG－Ⅰ相比，MPEG－Ⅱ支持更广的分辨率和比特率范围，将成为数字图像盘(DVD)和数字广播电视的压缩方式；另一种标准——MPEG－Ⅳ正在发展中，它可支持非常低的比特率数据流，如电视电话、视频邮件和电子报刊等。

MPEG压缩算法中包含两种基本技术：一种是基于16×16子块的运动补偿技术，用来减少帧序列的时域冗余；另一种是基于DCT的压缩，用于减少帧序列的空域冗余，在帧内压缩及帧间预测中均使用了DCT变换。时间压缩依赖于相邻图像之间的相似性，并利用了预测和运动补偿技术。而空间压缩依赖于图像各小范围内的冗余，并以DCT变换、量化和熵编码技术为基础，预测得到的图像是前一图像进行运动补偿后得到的。运动余量是对每一个宏像块进行计算得出来的，运动矢量适用于宏像块中的所有4个亮度像块，用于2个色度像块的运动矢量是从亮度矢量计算出来的。

运动补偿算法是当前视频图像压缩技术中使用最普遍的方法之一、帧序列的相邻画面之间的运动部分是具有连续性的，当前画面上的图像可以看成是前面某时刻画面上图像的位移，位移的幅度值和方向在画面各处可以不同。利用运动位移信息与前面某时刻的图像信息对当前画面图像进行预测的方法，称为前向预测；反之，根据某时刻的图像与位移信息预测该时刻之前的图像，称为后向预测。MPEG的运动补偿预测方法将画面分成若干个16×16的子图像块(称为补偿单元或宏块)，并根据一定的条件分别进行帧内预测、前向预测、后向预测及平均预测。以插补方法补偿运动信息是提高视频压缩比的最有效措施之一。在时域中插补运动补偿是一种多分辨率压缩技术。例如，以1/15秒或1/10秒时间间隔选取参考子图，对时域较低分辨率子图进行编码，通过低分辨子图及反映运动趋势的附加校正信息(运动矢量)进行插值，可得到满分辨率(帧率1/30秒)的视频信号。插值运动补偿也称为双向预测，因为它既利用了前面帧的信息，又利用了后面帧的信息。

MPEG压缩算法分三个层次完成压缩，即带宽压缩、匹配主观的有损失压缩和最后一层的无损失压缩。第一层主要是源分解率、目标比特率匹配以及降低色度的分解率，可达

到主观上满意的程度；第二层即压缩算法本身，其主要功能是利用波形分析和主观适配的量化来去掉空间冗余和时间冗余，在这个层次压缩是有损失的；第三层的功能是通过把固定长度和可变长度编码进行句法组合，无损失地把信息变换到比特流中去。

MPEG 图像编码包含三个成分：I 帧、P 帧和 B 帧。MPEG 编码过程中，一些图像压缩成 I 帧，一些压缩成 P 帧，另一些压缩成 B 帧。I 帧压缩可以得到 6：1 的压缩比，而不产生任何可觉察的模糊现象。I 帧压缩的同时使用 P 帧压缩，可以达到更高的压缩比而不出现无可觉察的模糊现象。B 帧压缩可以达到 200：1 的压缩比，其文件尺寸一般为 I 帧压缩尺寸的 15%，不到 P 帧压缩尺寸的一半。

I 帧压缩采用基准帧模式，只提供帧内压缩，即把帧图像压缩到 I 帧时，仅仅考虑了帧内的图像。I 帧压缩能够去掉图像的空间冗余度，但是不能除去帧间冗余度。帧内压缩基于离散余弦变换(DCT)，类似于 JPEG 和 H. 261 图像中使用的 DCT 压缩标准。

P 帧采用预测编码，利用相邻帧的一般统计信息进行预测。也就是说，它考虑运动特性，提供帧间编码，P 帧预测当前帧与前面最近的 I 帧或 P 帧的差别。B 帧为双向帧间编码，它从前面和后面的 I 帧或 P 帧中提取数据。B 帧基于当前帧与前一帧和后一帧图像之间的差别进行压缩。P 帧和 B 帧可去掉时间冗余度，它们要求计算机要具备更强的功能。有些压缩器不能产生 B 帧，甚至连 P 帧也不能产生，这样的图像压缩结果将会出现很明显的间断。

## 【习　题】

1. 利用 10.2.2 节中介绍的行程编码对图 10.5 所示图像进行行程编码。

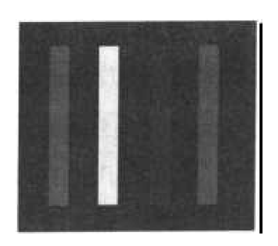


图 10.5　待编码图像

2. 利用 MATLAB 实现图像的算术编码。
3. 利用 MATLAB 的方程组求解函数实现线性预测编码。

# 第十一章 图像分析和理解

**本章要点：**

- ★ 边缘检测方法
- ★ 区域分割技术
- ★ 形状分析
- ★ 区域操作

## 11.1　边缘检测方法

### 11.1.1　边缘检测概述

在图像处理技术中，许多场合都要求用计算机进行图像描述并对图像进行分析和理解。例如，对于大规模集成电路的自动检测过程，要求对图像进行分析以得到芯片图像中有关疵点的描述；对于医学癌细胞识别来说，要求能够从显微镜中得到有关癌细胞形状的描述。诸如此类的图像处理应用领域都要用到图像的分析和理解技术，尤其是机器人视觉领域，图像的分析理解更是其研究的核心内容。

从图像的分析和理解来说，最基本的两个内容就是图像的分割和区域描述。图像分割是指将图像中有意义的对象与其背景分离，并把这些对象按照不同的含义分割开来，也就是说，图像分割就是将图像中具有不同含义的对象提取出来。区域描述是对对象本身及对象间关系的描述，使之具有某种指定的数学或符号表达形式，使计算机能够理解具体对象的具体含义。

图像分割可以分为两种：一种是基于边界的分割技术；另一种是基于区域的分割技术。边缘检测技术是所有基于边界分割的图像分析方法的第一步，检测出边缘的图像就可以进行特征提取和形状分析了。两个具有不同灰度的相邻区域间总是存在着一定的边界，这个边界就是灰度值不连续的结果。为了计算方便起见，通常选择一阶和二阶导数来检测边界。图 11.1 给出了图像边界所对应的一阶导数和二阶导数曲线，从图中可以看出，利用求导方法可以很方便地检测到灰度值的不连续效果。

边缘的检测可以借助空域微分算子（实际上是微分算子的差分近似）利用卷积来实现。常用的微分算子有梯度算子和拉普拉斯算子等，这些算子不但可以检测图像的二维边缘，还可以检测图像序列的三维边缘。下面我们将对这些算子进行一一介绍。

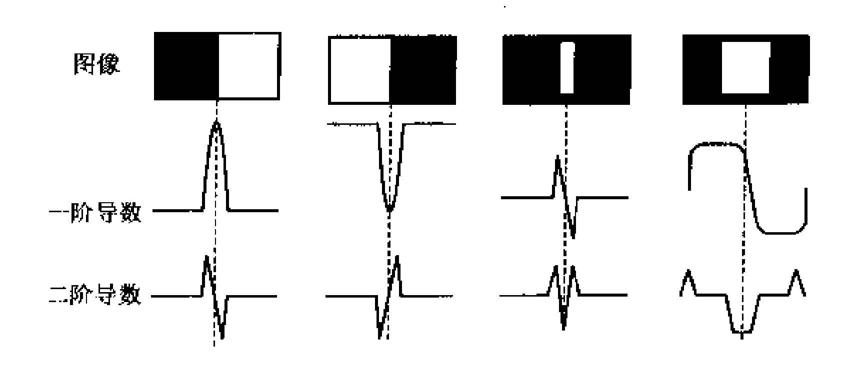



图 11.1 图像边缘及相应导数

### 11.1.2 梯度算子

梯度对应于一阶导数，相应的梯度算子就对应于一阶导数算子。对于一个连续函数 $f(x,y)$，其在 $(x,y)$ 处的梯度定义如下：

$$\nabla f(x,y) = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{bmatrix}^{\mathrm{T}} = [G_x \quad G_y]^{\mathrm{T}} \tag{11.1}$$

梯度是一个矢量，其幅值和相位分别为

$$|\nabla f| = [G_x^2 + G_y^2]^{1/2} \tag{11.2}$$

$$\phi(x,y) = \arctan\left(\frac{G_y}{G_x}\right) \tag{11.3}$$

式(11.1)～(11.3)中的偏导数需要对每一个像素位置进行计算，在实际应用中常常采用小型模板利用卷积运算来近似，$G_x$ 和 $G_y$ 各自使用一个模板。人们已经提出了许多不同大小、不同系数的模板，最简单的是 Roberts 算子，其模板如下：

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{11.4}$$

较为复杂的常用模板有 Prewitt 算子和 Sobel 算子，其模板分别如式(11.5)和(11.6)所示。

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \tag{11.5}$$

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{11.6}$$

图 11.2 分别给出了利用这三个算子进行边缘检测的不同效果，从图中可以看出，在这三种模板中，Sobel 算子的检测效果最好。

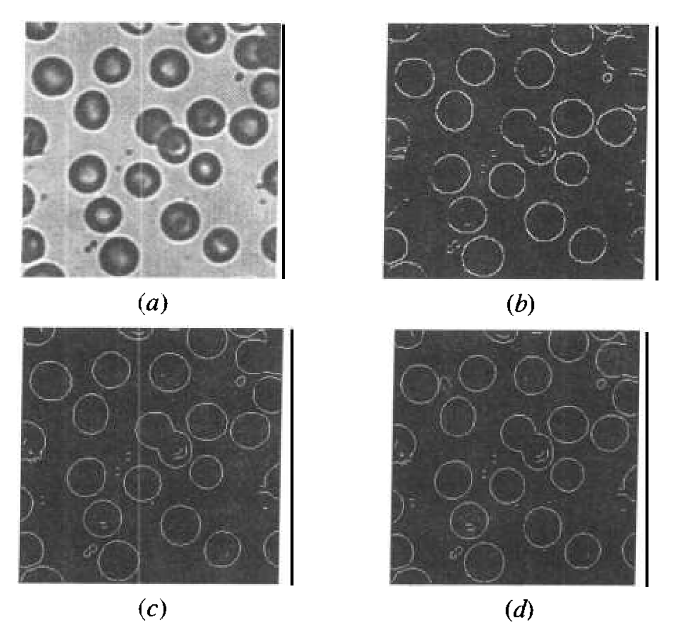


图 11.2 不同微分算子的边缘检测效果

($a$) 原始图像；($b$) Roberts 算子检测；($c$) Prewitt 算子检测；($d$) Sobel 算子检测

### 11.1.3 拉普拉斯算子

拉普拉斯算子(Laplacian)是一种二阶导数算子。对一个连续函数 $f(x,y)$，其在$(x,y)$处的拉普拉斯算子定义如下：

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{11.7}$$

在图像处理过程中，函数的拉普拉斯算子也是借助模板来实现的。这里对模板有一个基本要求：模板中心的系数为正，其余相邻系数为负，所有系数的和应该为零。常用的两种模板如式(11.8)所示。

$$\underset{\text{拉普拉斯算子一}}{\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}} \qquad \underset{\text{拉普拉斯算子二}}{\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}} \tag{11.8}$$

图 11.3 给出了对图 11.2($a$)利用以上两种拉普拉斯算子进行边缘检测的效果。从图中可以看出，拉普拉斯算子边缘检测方法常常产生双像素边界，另外该检测方法还对噪声比

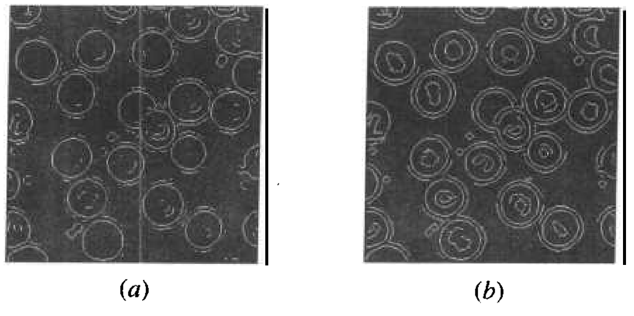


图 11.3 不同拉普拉斯算子的边缘检测效果

($a$) 拉普拉斯算子一；($b$) 拉普拉斯算子二

较敏感，而且不能提供边缘方向信息，所以一般很少直接使用拉普拉斯算子进行边缘检测，而是常用它来判断边缘像素是位于图像的明区还是暗区。

### 11.1.4　边缘连接方法

显然，利用微分算子对图像进行检测所得的边界常常会发生断裂现象。将边缘连接起来的方式主要有这样几种：邻域端点搜索、启发式搜索、曲线拟合和 Hough 变换。

邻域端点搜索主要是针对间隙较小的边界裂缝，通过搜索以断裂边界点为中心的 5×5 邻域找到其他边界端点，并填充必要的边界像素，从而将边界连接起来。这种方法对于图像边缘点较多的复杂图像会产生过渡分割现象，因而使用该连接时还须额外规定：两个端点只有在边缘强度和方向都相近的情况下才能进行连接。

启发式搜索首先建立一个可以在任意两端点 A 和 B 的连接路径上进行计算的函数，该边缘计算函数是该边界上各点减去方向偏差分量后的强度平均值。然后利用该计算函数对 A 的各个邻域点进行评价，确定哪一个像素可以作为走向 B 点的候选连接点，通常仅仅考虑那些位于朝向 B 方向上的像素点。最后在候选点中选择一个能够使 A 点到该点的边缘计算函数取值最大的点作为连接点，并作为下一次迭代过程的起点。如此迭代下去，直到最后连接到 B 点为止。当最終连接到 B 点时，还要将边界计算函数的取值与一个阈值进行比较，满足阈值条件的迭代结果才能够真正作为边界，否则将被抛弃。这种搜索方法对那些只有一个缺口，而又不能用一条直线连接断点的边界裂缝非常有效，但是对那些缺口较多、函数计算过程较困难的图像来说将会过于复杂。

如果图像的边界点分布非常稀疏，那么可以使用分段线性或高阶样条曲线来拟合这些边缘点，从而形成一条适用边界。具体的曲线拟合方法请读者参考有关书籍。

Hough 变换是利用图像的全局特征将边缘像素连接起来形成封闭边界的一种连接方法。假设需要从给定图像的 $n$ 个点中确定哪些点位于同一条直线上，那么可以将其看成是根据已知直线上的若干点来检测直线的问题。解决这一问题的一个直接方法就是先确定所有由任意两点决定的直线，然后从中找出接近具体直线的点集，这样做大约需要进行 $n^2+n^3$ 次运算才能完成，实际中是很难满足这样大的计算量要求的。利用 Hough 变换就可以用较少的计算量来解决这一问题。Hough 变换的基本思想是利用了点与直线的对偶性的特性，在图像空间中，所有过点 $(x,y)$ 的直线都满足以下方程：

$$y = kx + b \tag{11.9}$$

其中，$k$ 表示直线的斜率，$b$ 为截距。用极坐标表示式(11.9)为

$$\rho = x\cos\theta + y\sin\theta \tag{11.10}$$

其中，$(\rho,\theta)$ 定义了一个从原点到直线上最近点的向量，该向量与直线垂直。式(11.10)就称为直线的 Hough 变换。显然，$x-y$ 平面中的任意一条直线都与 $\rho-\theta$ 空间(称为参数空间)的一个点相对应，也就是说，$x-y$ 平面中的任意直线的 Hough 变换是参数空间中的一个点，在 $x-y$ 平面中，过点 $(x,y)$ 的直线有很多条，每一条都对应参数空间的一个点，此时可以将式(11.10)中的 $x$ 和 $y$ 看成是参数空间中的常数，那么 $x-y$ 平面中过点 $(x,y)$ 的直线所对应的点就在参数空间中形成了一条正弦曲线，也就是说，$x-y$ 平面上的点对应于参数空间中的一条正弦曲线。由于 $x-y$ 平面中的直线上的各个边缘点都满足参数相同(假设为 $\rho_0$ 和 $\theta_0$)的等式(11.10)，所以 $x-y$ 空间中所有边缘点对应的正弦曲线都相交于点

$(\rho_0,\theta_0)$。为了找出这些边缘点构成的直线，可以建立一个位于参数空间中的直方图，对于每一个边缘点，给参数空间中所有与之对应的正弦曲线的直方图方格一个增量。于是，当所有边缘点都经过这种处理后，包含$(\rho_0,\theta\#-0)$的方格将具有局部最大值，通过对参数空间的直方图进行局部最大值搜索就可以获得边界直线的参数。

Hough 变换的主要优点是受噪声和曲线间断的影响较小。利用 Hough 变换除了可以进行边界连接之外，还可以用来直接检测某些已知形状的目标。

### 11.1.5　边缘检测的 MATLAB 实现方法

可以使用 MATLAB 图像处理工具箱中的 edge 函数利用以上算子来检测边缘。edge 函数提供许多微分算子模板，对于某些模板可以指定其是对水平边缘还是对垂直边缘(或者二者都有)敏感(即主要检测是水平边缘还是垂直边缘)。edge 函数在检测边缘时可以指定一个灰度阈值，只有满足这个阈值条件的点才视为边界点。edge 函数的基本调用格式如下：

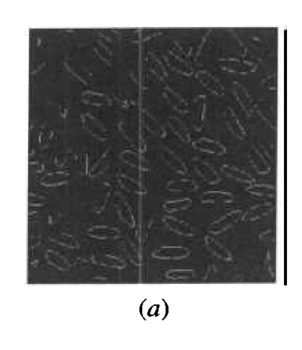
```
Bw=edge(I,'type',parmeter,…)
```

其中，I 表示输入图像，type 表示使用的算子类型，parmeter 则是与具体算子有关的参数。例如，以下语句将指定 Sobel 算子检测到的边界的域值和敏感方向：

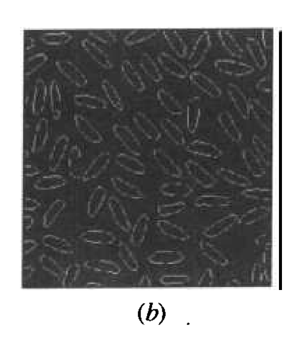
```
BW = edge(I,'sobel',THRESH,DIRECTION)
```

其中，DIRECTION 可以取值 horizontal、vertical 或 both。edge 函数返回一个用数值 1 代表找到的边缘，用数值 0 代表其他像素的二进制图像。edge 函数提供的最有效的边缘检测方法是 Canny 方法。Canny 方法也使用拉普拉斯算子，该方法与其他边缘检测方法的不同之处在于，它使用两种不同的阈值分别检测强边缘和弱边缘，并且仅当弱边缘与强边缘相连时才将弱边缘包含在输出图像中，因此这种方法较其他方法而言不容易被噪声“填充”，更容易检查出真正的弱边缘。以下代码给出了针对同一幅图像的 Sobel 和 Canny 两种边缘检测器的不同效果，如图 11.4 所示。从图 11.4 中可以看出 Canny 方法的优越之处。

```
I = imread('rice.tif');
BW1 = edge(I,'sobel');
BW2 = edge(I,'canny');
subplot(1,2,1),imshow(BW1)
subplot(1,2,2),imshow(BW2)
```



(*a*)



(*b*)

图 11.4　sobel 算子和 Canny 方法边缘检测效果
(*a*) sobel 算子；(*b*) Canny 方法

# 11.2 区域分割技术

## 11.2.1 阈值分割

阈值分割是一种最常见的直接检测区域的分割方法。利用阈值方法来分割灰度图像时都是基于一定的图像模型的，也就是说，要对图像进行一定的假设，一般都假设图像由具有单峰灰度分布的目标和背景组成，目标或背景内部的相邻像素间的灰度值是高度相关的，但是在目标和背景交界处的像素在灰度值上有很大的差别。如果图像满足这个条件，那么其灰度直方图基本上可以看成是由目标和背景两个单峰值方图混合形成的。如果这两个单峰直方图分布情况接近并且均值相差较远、均方差较小，那么混合直方图应该是双峰的，这一类图像用阈值方法就可以很好地进行分割。

对于只有两个灰度值的图像(二进制图像)，阈值分割步骤如下：首先对灰度值分布在$g_{min}$和$g_{max}$之间的图像指定一个灰度阈值$T(g_{min}<T<g_{max})$，然后将图像中每个像素的灰度值与阈值相比较，并将对应的像素根据比较结果分为两类：像素灰度值大于(或大于等于)阈值的分为一类，其余的分为一类。这两类像素一般对应于图像中的两类区域。如果图像中包含多个灰度值区域，那么可以选择一系列的阈值来分割各个像素。

显然，阈值的选取是阈值分割最关键的地方。假设$f(x,y)$是像素$(x,y)$的灰度值，$p(x,y)$表示该像素点邻域的某种特征，那么可以将阈值$T$写成如下形式：

$$T = T[x,y,f(x,y),g(x,y)] \tag{11.11}$$

根据式(11.11)可以将阈值$T$分为三类：

■ 仅仅根据$f(x,y)$选取的阈值，这样所得的阈值仅与像素的本身性质有关；

■ 根据$f(x,y)$和$g(x,y)$选取的阈值，这样所得的阈值是与图像的局部性质有关的；

■ 除了$f(x,y)$和$g(x,y)$以外，阈值还与$x$和$y$有关，这样所得的阈值是与像素坐标有关的。

第一种阈值选取方法是最简单的选取方法，一般是根据图像的直方图来进行的。如果图像的灰度直方图是一个双峰直方图，那么选择两峰之间的谷值就可以作为阈值来分割目标和背景。可以利用imregionalmin函数来搜索直方图中的谷值，然后进行阈值分割。

在实际应用中，图像常常因为受到噪声影响而使原本分离的直方图双峰发生重叠，在这种情况下很难直接找到谷值作为阈值进行分割。虽然此时直方图基本上是单峰的，但是峰的某一侧通常要比另一侧陡峭。此时，基于局部特征的阈值选取方法就可以解决阈值选取中存在的问题。常用的方法有直方图变换、利用过渡区进行阈值选取等。直方图变换的基本思想是利用像素邻域的性质对原来的直方图进行变换，从而得到一个新的直方图。这个新的直方图通常比原直方图具有更深的谷值，或者将原来的谷值转换为现在的峰值。常用的像素邻域性质一般是像素的梯度值，可以借助梯度算子作用于像素邻域来实现。由于一次只能利用一个灰度级的梯度算子，因而这种方法很容易受到噪声或图像中的非目标区域的影响，这些影响通常会使变换后的直方图具有虚假的极值，干扰阈值的正确选取。基于过渡区的选取方法同时利用了不同灰度值像素的梯度信息，所以其抗干扰性能较好。过渡区是介于图像中目标和背景间的一类区域，可以借助图像有效平均梯度计算和图像灰度

剪切操作来确定。假设像素$(i,j)$的灰度值为$f(i,j)$，$g(i,j)$表示$f(i,j)$的梯度值，那么图像的有效平均梯度定义如下：

$$EAG=\frac{\sum_{i,j\in Z}g(i,j)}{\sum_{i,j\in Z}p(i,j)}=\frac{TG}{TP} \tag{11.12}$$

其中，$TG$表示梯度图的总梯度值，$TP$表示非零梯度像素的总数，$p(i,j)$的定义如下：

$$p(i,j)=\begin{cases}1 & g(i,j)>0\\ 0 & g(i,j)=0\end{cases}$$

为了减少各种干扰对阈值的影响，需要用到一种特殊的剪切变换：高帽剪切和低帽剪切。假设$L$为剪切值，那么高帽剪切图像为

$$f_{\text{high}}(i,j)=\begin{cases}L & f(i,j)\geqslant L\\ f(i,j) & f(i,j)<L\end{cases} \tag{11.13}$$

低帽剪切图像为

$$f_{\text{low}}(i,j)=\begin{cases}L & f(i,j)<L\\ f(i,j) & f(i,j)\geqslant L\end{cases} \tag{11.14}$$

如果对这样剪切后的图像进行梯度变换，那么其梯度函数必定与$L$有关，因而$EAG$也就与$L$有关，分别记为$EAG_{\text{high}}(L)$和$EAG_{\text{low}}(L)$。典型的$EAG_{\text{high}}(L)$和$EAG_{\text{low}}(L)$曲线都只有一个峰值，分别记为$L_{\text{high}}$和$L_{\text{low}}$。这两个极值就限定了过渡区的灰度范围，它们具有三个非常重要的性质：对于每个过渡区，$L_{\text{high}}$和$L_{\text{low}}$都惟一存在；$L_{\text{high}}$和$L_{\text{low}}$所对应的灰度值具有明显的像素特性区别能力；对同一个过渡区，实际图像中的$L_{\text{high}}$总是大于$L_{\text{low}}$。

根据$L_{\text{high}}$和$L_{\text{low}}$得到的过渡区是一个环绕目标边界的带状区，所以可以将这个区域进行细化或骨架化，从而得到真正的边界。

最后一种依赖于像素坐标的阈值选取方法主要是在图像中背景亮度不一致的情况下进行的操作，其基本思想是：首先将图像分解为一系列子图像(可以有重叠部分)，然后为每一个子图像选择一个阈值(可以采用最简单的阈值选取方法)，最后对这些子图像的阈值进行插值就可以得到最终的图像阈值。比较常用的依赖于坐标的阈值选取方法的基本步骤如下：

(1) 将整幅图像分为一系列相互之间由50%重叠的子图像。

(2) 做每个子图像的直方图。

(3) 检测各个子图像的直方图是否为双峰的，如果是，则选择谷值作为子图像的阈值，否则不予处理。

(4) 对能够得到的阈值进行插值计算，得出所有子图像的阈值。

(5) 对所有子图像的阈值再进行插值，从而得到一个最终的阈值。

(6) 图像的进一步分析和理解。

### 11.2.2 区域生长

以上介绍的阈值方法是对整幅图像做一次性的分割处理，是一种并行处理技术，而区域生长则是一种将处理过程分解为多个顺序步骤的串行方法，每一个后续步骤都要依据前面步骤的结果进行判断，一般采用基于图像灰度特性的准则作为判断依据。

区域生长的基本思想是将具有相似性质的像素集合起来构成区域。首先对每一个需要分割的区域寻找一个种子像素作为生长的起点，然后将种子像素周围邻域中与种子像素具有相同或相似性质(实现指定的生长或相似准则)的像素合并到种子像素所在的区域中，将这些新像素作为新的种子像素继续进行上述过程，直到再没有符合条件的像素被包括进来为止，这样一个区域就长成了。

在实际应用区域生长法时需要解决三个问题：一个是如何选择一组能正确代表所需区域的种子像素；二是如何确定在生长过程中能将相邻像素包括进来的准则；三是如何制定生长停止条件或规则。

第一个问题通常可以根据具体图像的特点来选取种子像素。例如，在红外图像检测技术中，通常目标的辐射都比较大，所以可以选择图像中最亮的像素作为种子像素。如果没有图像的先验知识，那么可以借助生长准则对像素进行相应计算。如果计算结果可以看出聚类情况，那么可以选择聚类中心作为种子像素。

第二个问题的解决不但依赖于具体问题的特征，还与图像的数据类型有关。如果图像是 RGB 图像，那么如果使用单色准则就会影响分割结果。另外，还需要考虑像素间的连通性，否则将会出现无意义的分割结果。

一般的生长过程都是在进行到没有满足生长准则条件的像素时就停止，但是常用的灰度、纹理和彩色准则等大都是基于图像的局部特性的，而与生成过程没有直接的关系。在这种情况下就需要对分割结果建立一定的准则。

### 11.2.3　分裂合并

分裂合并分割方法的基本思想与区域生长正好相反，是从整幅图像开始不断分裂得到各个区域。最常用的分裂合并方法就是四叉树分解方法。

令 $R$ 代表整个正方形的图像区域，可以将 $R$ 连续地分裂为越来越小的 1/4 正方形子区域 $R_i$，并保证每一个子区域符合预先指定的分裂准则(参见图 11.5)。换句话说，就是对符合分裂准则的区域不断地进行四等分，以此类推，直到 $R_i$ 为单个像素为止，这样就会生成如图 11.5 所示的四叉树。

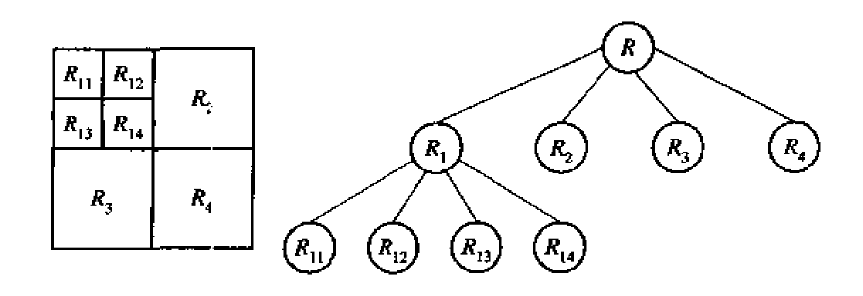



图 11.5　图像的四叉树分解法示意图

事实上，如果只对图像进行分裂而不合并，那么最后可能导致相邻两个区域具有相同性质，但是却被划分为两个区域，所以在每一次分裂操作后还允许子区域按照一定的准则进行合并。

### 11.2.4　区域分割的 MATLAB 实现方法

阈值分割是一种较为简单的区域分割技术，可以通过调用形态学操作函数 imtophat 和 imbothat 函数来实现高帽和低帽变换，然后绘制经过变换后的图像的直方图，最后调用 imregionalmin 函数寻找直方图中的谷值，将其作为阈值对图像进行分割。区域生长分割方法的实际应用需要根据具体图像的具体特征来确定种子像素和生长及停止准则，其通用性不是很强，这里不作介绍，有兴趣的读者可以参考有关书籍了解这种分割方法的实现方法。本书将主要介绍最为有效而常用的四叉树分解实现方法。

可以调用图像处理工具箱中的 qtdecomp 函数来实现四叉树分解。这个函数首先将图像划分为相等大小的正方形块，然后对每一个块进行测试，观察它们是否与标准具有相同性。如果某个块符合标准，那么就不对其进行进一步的分割；如果不符合这种规律，那么将该块继续划分为四块，将测试标准应用于其他块。这个过程将会一直重复直至每一个块都符合这个标准。分解的结果可能会包含许多不同大小的块。例如，假设用户希望对一幅 128×128 的灰度图像进行四叉树分解，第一步是将图像划分为 64×64 的块，然后对每一个块应用例如以下的测试标准：

```
max(block(:)) - min(block(:)) <= 0.2
```

如果其中一个块符合这个标准，则这个块将不会再进行分解，分解完成后仍然是一个 64×64 的块；如果块不符合这个标准，那么将被分解为四个 32×32 的块，然后再次对这些块应用相同的测试标准。不符合标准的块将被再次分解为四个 16×16 的块，以此类推，直至所有的块都通过测试，有些块可能只有 1×1 的大小。

qtdecomp 的基本调用方法如下：

```
S = qtdecomp (I,THRESHOLD, [MINDIM MAXDIM])
```

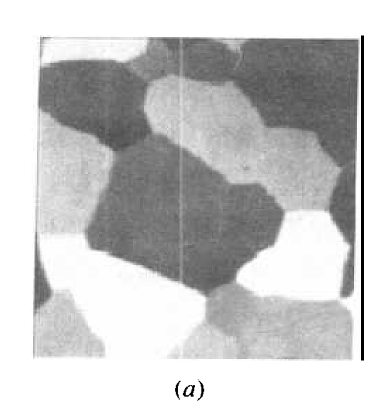
其中，I 是输入图像。THRESHOLD 是一个可选参数，如果某个子区域中的最大像素值减去 MINDIM 后的结果大于由 THRESHOLD 指定的阈值，那么分解继续进行，否则停止分解并返回。[MINDIM MAXDIM]也是可选参数，用来指定最终分解得到的子区域大小，如果子区域不在这个区域中，那么分解继续进行，否则返回。返回值 S 是一个稀疏矩阵，其非零元素的位置对应于块的左上角坐标，每一个非零元素的数值代表块的大小。S 的大小与 I 相同。

⚠ **注意：**

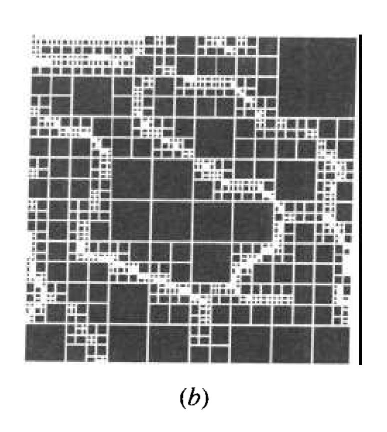
*无论 I 是何种类型的，阈值都是一个指定在 0 到 1 之间的数值。如果 I 是 uint8 类型的，那么指定的阈值将会乘以 255 作为真实的阈值使用；如果 I 是 uint16 类型的，那么使用的阈值将会乘以 65535。*

⚠

图 11.6 给出了一幅原始图像及其四叉树分解结果。每一个正方形代表一个同性块，白线表示块之间的边界。从图中可以看出，图像灰度变化越大，子区域的面积越小，分割结果也就越精确。



(a)



(b)

图 11.6　图像及其四叉树分解结果
(a) 原图像；(b) 四叉树分解结果

# 11.3 形 状 分 析

## 11.3.1　启发式搜索

形状是图像分类中最常用的概念。形状分析法主要有：启发式方法、变换方法、细化(骨架化)等。对于形状问题，最大的困难是不知道在数学上如何对它下定义，所以很多情况下都采用启发式方法。启发式搜索又叫作非数学方法，通常所用的启发式方法是取对象周长的平方与对象面积的比值来描述对象的形状，因为这个比值可以明显地区分出长形物体和圆形物体：对于长形物体，该比值一般较大；对于圆形物体，该比值较小。

在 MATLAB 中，可以利用形态学操作函数 bwarea 计算根据图像分割得到的目标面积，然后计算边界长度作为目标周长，最后利用启发式搜索方法进行物体的识别。显然，这种启发式方法用于监测或识别机器零件是非常有效的，但是这种方法没有严格的理论依据，找不到一般规律，所以不具有通用性，不宜推广。

## 11.3.2　变换方法

图像的变换方法很多，应用最广的一种是傅立叶变换方法，就是沿着图像的轮廓作傅立叶变换。变换方法中的傅立叶变换系数有好几种定义，应用最广的有两种。第一种方法说明如下：对于图像轮廓 $r$，假设沿 $r$ 行进的时间函数为 $\varphi(t)$，行进到轮廓上任一点的坐标为$[x(t),y(t)]$，则傅立叶描述式为

$$A_k = \frac{1}{2\pi}\int 2\pi\varphi(t)\mathrm{e}^{-\mathrm{j}nt}\mathrm{d}t \tag{11.15}$$

从式(11.15)中可以看出，如果轮廓 $r$ 为一个圆，那么其傅立叶描述式的系数为 0，其表达式非常简单。但是对于不连续的图形(例如，多边形)，如果用一个有限项的和来近似，那么系数衰减就很慢，必须用很多谐波才能近似，使用不方便。

第二种方法对图像的每一个点不是采用角度，而是采用长度作为参数，用复数形式可以表示为 $x(l)+\mathrm{i}y(l)$，起始点为 $l=0$，全长为 $L$，其傅立叶描述式为

$$a_n = \frac{1}{L}\int u(l)d^{-j\frac{2\pi}{L}nl}\mathrm{d}l \tag{11.16}$$

其中

$$u(l) = \sum_{n=-\infty}^{\infty} a_n e^{jn\frac{2\pi}{L}l}$$

式中，$a_n$ 为傅立叶描述式。由于其衰减很快，所以可取有限项的和来近似上式：

$$u(l) = \sum_{n=-M}^{M} a_n e^{jn\frac{2\pi}{L}l} \tag{11.17}$$

用以上方法求出 $a_n$ 以后，就可以用最佳曲线匹配(即模板匹配)来识别图像轮廓了。例如，假设有两个轮廓曲线：一个是标准图像轮廓曲线 $A$，另一个是待识别的轮廓曲线 $B$，对两个曲线求傅立叶描述式(曲线 $A$ 的傅立叶描述式系数为 $a_n$，曲线 $B$ 的傅立叶描述式系数为 $b_n$)，然后检测两条曲线是否匹配，这时可以用傅立叶系数间的距离代表匹配程度：

$$d(A,B) = \left[\sum_{n=-M}^{M} |a_n - b_n|^2\right]^{1/2} \tag{11.18}$$

由于计算 $A$ 和 $B$ 的傅立叶描述式时的起始点有可能不同，因而可能得到不同的结论。为此，假设 $A$ 为输入图像，那么可以将 $B$ 视为训练采样进行存储；若要进行匹配，则可通过调整 $b_n$ 对起始点进行调整。这就要对 $b_n$ 进行变换，将其与一个变量 $Se^{j(n\alpha+\phi)}$ 相乘，其中 $S$ 为比例系数，$\phi$ 为旋转变换，然后求最小距离：

$$\min = \left[\sum_{n=-M}^{M} |a_n - Se^{j(n\alpha+\phi)}b_n|^2\right]^{1/2} \tag{11.19}$$

这就是变换模板的方法，即先求出模板与输入图像之间的最小距离，然后再确定其匹配程度。采用这种方法时，如果对象轮廓是一个圆，那么零次谐波系数 $a_0$ 恰好是圆心坐标，一次谐波系数 $a_1$ 就是圆的半径，其它项都为零，所以其形式较为简单，用这种方法进行机器零件的识别也很有效。事实上对图像作形状分析时大多是采用傅立叶描述式，用它可以识别机器零部件的形状、飞机的形状、心脏和细胞的形状等。形状分析的最大难点在于当对象形状有突变时需要用很多谐波来描述这种变化，如果两幅图像的差别很小，那么就要使用很多谐波才能区分两幅图像。

### 11.3.3 细化(骨架化)

细化方法就是通过细化用骨架来代表对象的形状。所谓细化，就是把输入的具有一定宽度的图像轮廓用逐次去掉边缘点的方法最终变为宽度仅为一个像素的骨架。例如，指纹识别系统就是利用骨架作为形状分析对象，用骨架的斜率代表形状。细化的 MATLAB 实现方法在第六章中已经做过介绍，此处不再赘述。

### 11.3.4 MATLAB 图像分析实例

**例 11.1**：利用图像分割测试图 11.7(*a*)所示的图像中的微小结构。

第一步，将图像读入变量 I 中：

```
I = imread('pearlite.tif');
subplot(2,2,1), imshow(I)
```

第二步，调用 imcomplement 函数对图像求补(即反色)，然后调用 im2bw 函数对图像

进行阈值截取分割并显示分割结果(如图 11.7(*b*)所示)。这一步骤的主要目的是获取图像中的微小物体。

```
Ic = imcomplement(I);
BW = im2bw(Ic, graythresh(Ic));
subplot(2,2,2), imshow(Ic)
```

第三步，抽取原始图像中较大的物体。首先对图像进行形态关闭，然后进行形态开启操作，从而达到删除图像中某些小对象的目的(如图 11.7(*c*)所示)。这里选择一个半径为 6 个像素的圆盘形结构元素。

```
se = strel('disk', 6);
BWc = imclose(BW, se);
BWco = imopen(BWc, se);
subplot(2,2,3), imshow(BWco)
```

经过以上步骤，我们已经得到了一幅包含小对象的图像和一幅包含大对象的图像，下面对这两幅图像进行逻辑“与”操作，从而得到原始图像中所有的小对象。最终的检测结果如图 11.7(*d*)所示。

```
mask = BW & BWco;
subplot(2,2,4),imshow(mask)
```

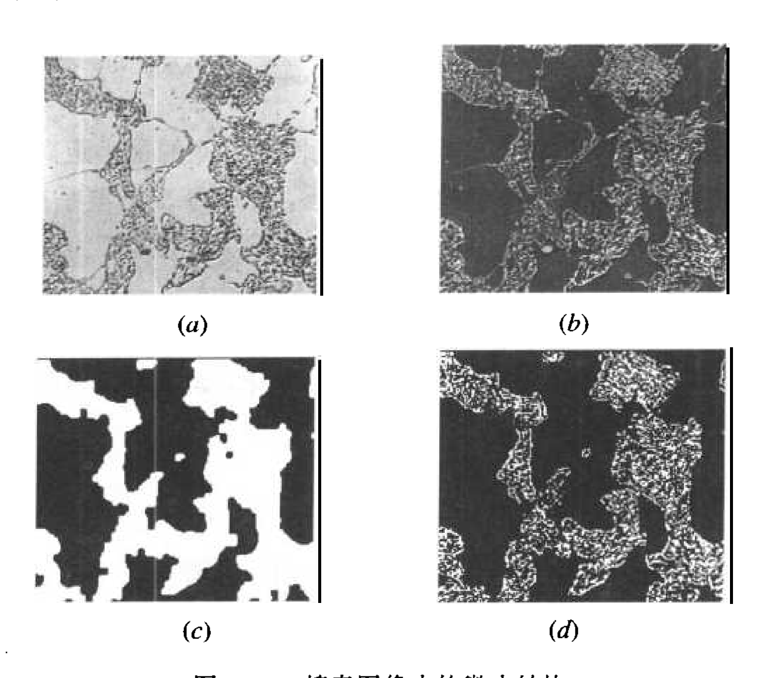


图 11.7　搜索图像中的微小结构

(*a*) 原图像；(*b*) 阀值截取分割后；(*c*) 对小图像进行删除后；(*d*) 检测结果

**例 11.2**：测试图 11.8(*a*)所示图像中相互接触的对象。

首先将图像读入变量 I 中：

```
afm = imread('afmsurf.tif');
subplot(2,3,1), imshow(afm)
```

由于图像中各个对象的灰度比较接近，因而进行图像分割之前应先将对比度调至最大。进行这种对比度调节最好的方法就是综合使用高帽变换和低帽变换。这两种变换都是

基于图像形态操作的，高帽变换实际上是图像开启结果与原始图像之差，低帽变换是图像关闭操作与原始图像之差。此处使用一个半径为 15 个像素的圆盘形结构元素进行图像的形态操作，高帽变换的结果如图 11.8(*b*)所示，低帽变换结果如图 11.8(*c*)所示。

```
se = strel('disk', 15);
Itop = imtophat(afm, se);
Ibot = imbothat(afm, se);
subplot(2,3,2), imshow(Itop, []),
subplot(2,3,3), imshow(Ibot, []),
```

从图 11.8(*b*)和 11.8(*c*)可以看出，高帽变换体现了原始图像中的灰度峰值，而低帽变换则体现了原始图像中的灰度谷值，为了能够找到最准确的阈值进行分割，首先将高帽变换结果与原始图像相加后再与低帽变换结果相减，从而得到最大对比度的图像。最后为了便于观察将图像反色(如图 11.8(*d*)所示)。

```
Ienhance = imsubtract(imadd(Itop, afm), Ibot);
Iec = imcomplement(Ienhance);
```

下面就可以调用 imextendedmin 函数来搜索图 11.8(*d*)的谷值，然后调用 imimposemin 函数将图像中所有谷值像素设置为 0(即为黑色，参见图 11.8(*e*))：

```
Iemin = imextendedmin(Iec, 22);
Iimpose = imimposemin(Iec, Iemin);
subplot(2,3,4),imshow(Iimpose)
```

下一步就可以调用分水岭分割函数对图 11.8(*e*)进行分割，为了观察方便，使用伪彩色显示技术显示分割结果：

```
wat = watershed(Iimpose);
rgb = label2rgb(wat);
subplot(2,3,5), imshow(rgb);
```

经过分割的图像就可以从中抽取感兴趣的对象特征了。在 MATLAB 中，图像的特征提取是通过函数 regionprops 进行的，该函数的调用格式如下：

```
STATS =regionprops (L,PROPERTIES)
```

调用过程中如果 PROPERTIES 为字符串 basic，那么返回值 STATS 将仅仅包括区域面积、质心和边框；如果为字符串 all，那么 STATS 将包括这样一些信息：'Area'、'ConvexHull'、'EulerNumber'、'Centroid'、'ConvexImage'、'Extrema'、'BoundingBox'、'ConvexArea'、'EquivDiameter'、'SubarrayIdx'、'Image'、'Solidity'、'MajorAxisLength'、'PixelList'、'Extent'、'MinorAxisLength'、'PixelIdxList'、'FilledImage'、'Orientation'、'FilledArea'、'Eccentricity'。PROPERTIES 还可以是以上字符串的任意组合。例如，此处将抽取区域面积和区域方向两个特征，并将这两个特征分别看成是函数的自变量和值域进行绘制，结果如图 11.8(*f*)所示。

```
stats = regionprops(wat, 'Area', 'Orientation');
area = [stats(:).Area];
orient = [stats(:).Orientation];
subplot(2,3,6), plot(area, orient, 'b*')
```

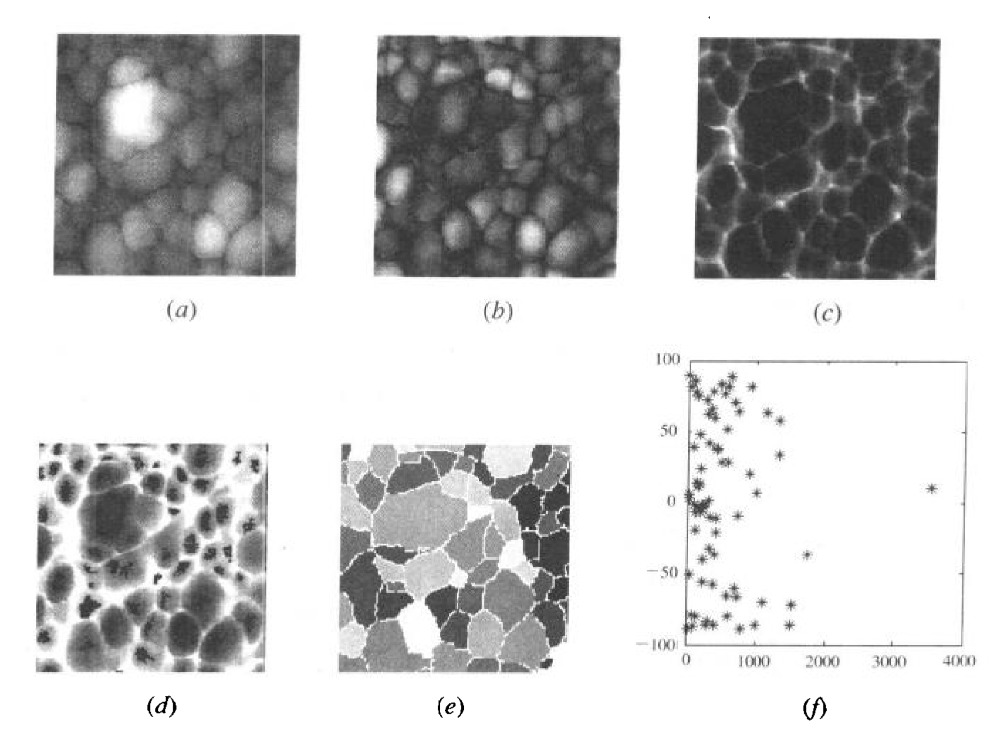


图 11.8　检测图像中相互接触的对象

(*a*) 原图像；(*b*) 高帽变换后；(*c*) 低帽变换后；

(*d*) 经反色等处理后；(*e*) 谷值搜索显示；(*f*) 特征信息

**例 11.3**：测试图 11.9(*a*)所示图像中星体的分布情况(即统计某一大小的星体所占的比例)。

首先将图像读入变量 I 中，然后将图像数据转换为双精度类型进行计算(这样才能够充分发挥 MATLAB 这种向量语言的长处)，并将图像的灰度范围调节为[0 1](即全部灰度范围)。

```
I = imread('ngc4024l.tif');
subplot(2,3,1),imshow(I)
gI = imadjust(im2double(I), [], [0 1]);
subplot(2,3,2),imshow(gI)
```

调节后的图像如图 11.9(*b*)所示。从图中可以看出，由于某些星体位于背景较亮的地方，因而显得很模糊。为此我们利用一个高帽变换消除图像背景中那些不一致的背景亮度：

```
se = strel('disk', 10);
topI = imtophat(gI, se);
subplot(2,3,3), imshow(topI)
```

下面就可以根据变换后的图像(图 11.9(*c*))来分析原始图像中的星体大小分布情况了。这里利用一个逐渐变大的结构元素来不断地进行图像形态开启，并统计开启后对象的剩余面积，通过绘制结构元素的大小和剩余面积的大小就可以计算出各种大小的星体在图像中占有的比例。

```
for counter = 0:20
    remain = imopen(topI, strel('disk', counter));
    surfarea(counter + 1) = sum(remain(:));
end
subplot(2,3,4), plot(surfarea, 'm - *'), grid on;
set(gca, 'xtick', [0 2 4 6 8 10 12 14 16 18 20]);
```

从图 11.9(*d*)所示的曲线可以看出，随着结构元素的增大，对象的剩余面积发生锐减。这是由于原始图像中含有较多的相同大小的星体的缘故。通过计算两次开启操作前、后的斜率(即一阶偏导)就可以估计出图像中相同大小星体所占的比例。

```
derivsurfarea = diff(surfarea);
subplot(2,3,5), plot(derivsurfarea, 'm - *')
```

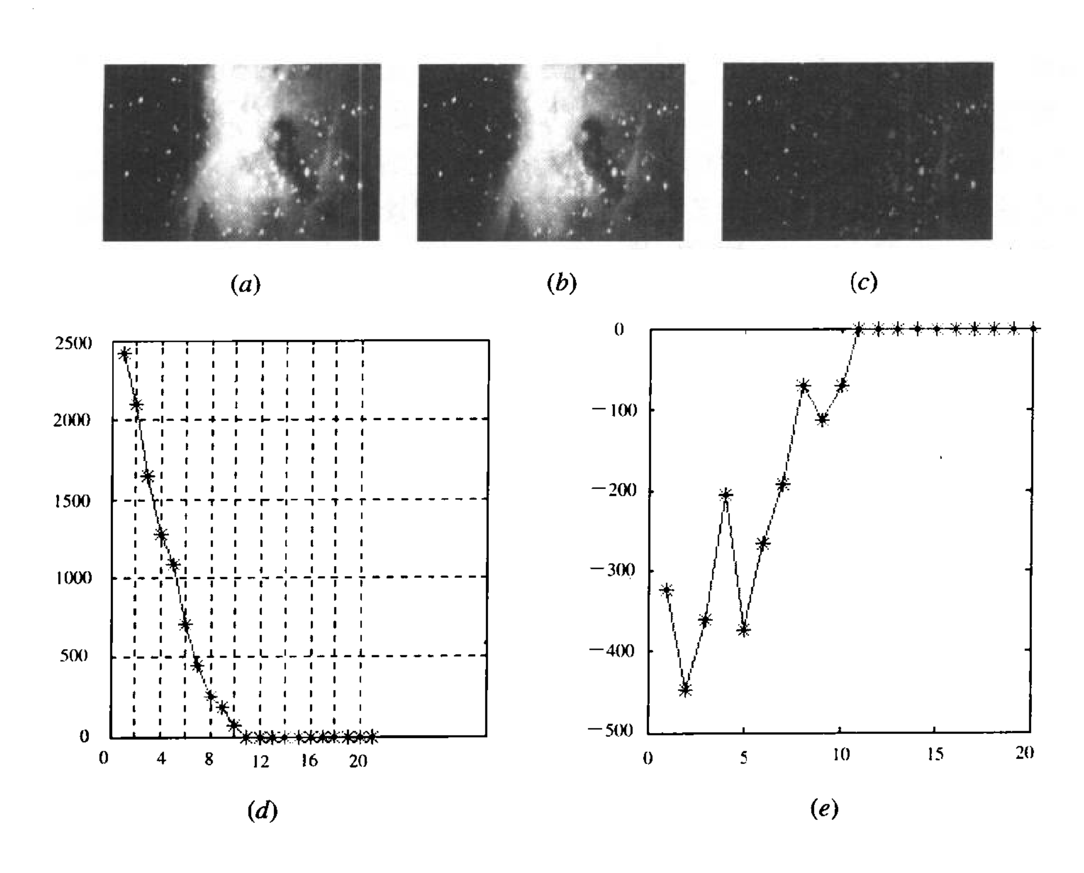



图 11.9　检测图像中星体的分布情况

(*a*) 原图像；(*b*) 数据类型调节后；(*c*) 高帽变换后；

(*d*) 剩余面积曲线；(*e*) 剩余面积变化曲线

从偏导后的图像(图 11.9(*e*))可以看出，原始图像中半径为 2 个像素的星体大约占了 500 个像素，半径为 5 个像素的星体大约占了 375 个像素，半径为 9 个像素的星体大约占了 125 个像素。注意，本例中使用的图像分析技术方法与以上两例稍有不同，本例中使用的方法不需要对图像进行精确的检测和分割。

# 11.4 区域操作

## 11.4.1 区域操作常用术语

感兴趣区域：图像中用户希望进行滤波或其他操作的部分。可以通过创建一个二进制掩模来定义感兴趣的区域。图像中可能存在多个感兴趣区域，区域本质上可以是各种图形（例如，环绕相邻像素的多边形），也可以定义为一个灰度范围。在后一种情况下，像素不一定是相邻的。

二进制掩模：一幅与待处理图像具有相同大小的二进制图像，该掩模图像中的数值 1 表示感兴趣区域中的像素，0 表示其他像素。

插值：根据图像像素间的关系估计某一位置处像素取值的过程。

掩模滤波：仅仅对图像中由二进制掩模定义的区域进行滤波的操作。在二进制掩模矩阵中，数值为 1 的像素返回滤波后的数值，为 0 的像素返回未滤波的数值。

区域填充：从区域边界开始对像素值进行插值的区域填充过程，这个过程可以用来隐藏图像中的对象，因为对象的像素值都将被原像素与背景区域像素的混合值代替。

区域滤波：对感兴趣的区域进行滤波的过程。例如，可以对图像的某一部分使用灰度调节滤波。

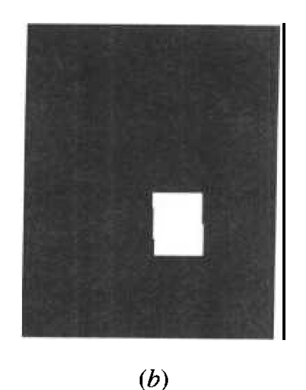
## 11.4.2 指定操作区域

在 MATLAB 中可以通过创建一个二进制掩模来定义感兴趣的区域。二进制掩模的创建方法有两种，一种是多边形选择方法，另外一种方法是调用 roicolor 函数创建任意的二进制掩模。

第一种方法通过调用 roipoly 函数来指定感兴趣的多边形区域。如果在调用 roipoly 函数时不指定任何输入参数，那么当鼠标位于显示在当前坐标轴上的图像中时将会变为十字形，然后可以通过使用鼠标点击多边形的端点来指定感兴趣的区域。端点选择完毕后点击【返回】命令，roipoly 函数将返回一个与输入图像相同大小的二进制矩阵，该矩阵中的数值 1 表示位于选中多边形之中的像素，0 代表其他位置的像素。下例说明了如何使用 roipoly 函数的交互式调用方法来创建一个二进制掩模。鼠标选择的区域边界如图11.10(*a*)所示(用方框显示)。



(*a*)



(*b*)

图 11.10　多边形区域
(*a*) 选择操作区域；
(*b*) 二进制掩模图像

```
I = imread('pout.tif');
imshow(I)
BW = roipoly;
```

roipoly 函数提供了一种创建二进制掩模的简单方法，其实，用户可以使用任意与被滤波图像大小相同的二进制图像作为掩模。例如，假设用户希望对灰度图像 I 进行滤波，并且仅对那些数值大于 0.5 的像素进行滤波，那么可以使用以下命令创建所需的掩模：

```
BW = (I > 0.5);
```

还可以使用 roicolor 函数根据颜色或灰度范围定义感兴趣的区域。

### 11.4.3　区域滤波

可以使用 roifilt2 函数对感兴趣的区域进行滤波。调用 roifilt2 函数时要指定一幅灰度图像、一个二进制掩模和一个滤波器。roifilt2 函数将对图像进行滤波并返回相应的数值，相对于掩模中数值为 1 的输出像素将返回滤波后的数值，而为 0 的输出像素则返回未滤波的数值，这种操作也称为掩模滤波。以下代码将使用图 11.10(*b*)所示的掩模来增强图11.10(*a*)中小女孩外衣上的商标图案：

```
h = fspecial('unsharp');
I2 = roifilt2(h,I,BW);
imshow(I)
figure, imshow(I2)
```



图 11.11　区域滤波效果

区域滤波结果如图 11.11 所示。

roifilt2 函数还能够使用自定义的操作函数对感兴趣区域进行处理。例如，以下代码将使用 imadjust 函数增加图像的局部亮度。此处使用的掩模是一个包含文本的二进制图像，因此输出图像也有文本显示在其中：

```
BW = imread('text.tif');
I = imread('cameraman.tif');
f = inline('imadjust(x,[],[],0.3)');
I2 = roifilt2(I,BW,f);
imshow(I2)
```



图 11.12　自定义区域滤波效果

自定义区域滤波效果如图 11.12 所示。

**注意：**

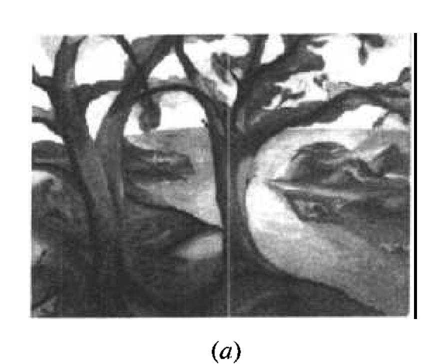
roifilt2 函数最适合于操作返回值与原始图像数据范围相同的图像，这是因为输出图像可以从输入图像中直接提取部分数据。各种滤波器操作都可能导致数据超出正常的数据范围(即双精度类型超过[0,1]，uint8 类型超过[0,255]，uint16 类型超过[0,65535])。

### 11.4.4　区域填充

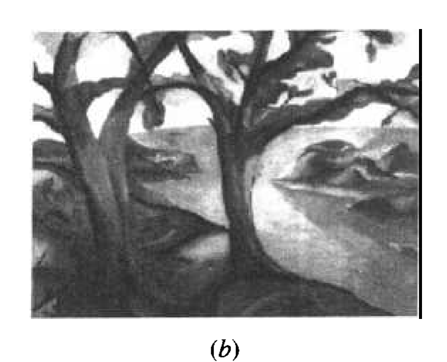
可以使用 roifill 函数从区域边缘开始插值来填充感兴趣的区域，这个函数在图像编辑(包括删除图像无关细节或人为痕迹)中非常有用。roifill 函数通过使用基于拉普拉斯方程的插值方法执行填充操作，给出区域的边界值，这个方法将产生最光滑的填充效果。和

roipoly 函数一样，roifill 函数可以使用鼠标来定义感兴趣的区域，选择完毕后，roifill 函数将返回一幅已经填充了所选区域的图像。以下代码使用 roifill 函数来修改树木图像(如图 11.13(*a*)所示)，被选区域的边框在原始图像中用多边形(◇)表示。

```
load trees
I = ind2gray(X,map);
subplot(1,2,1),imshow(I)
I2 = roifill;
subplot(1,2,2),imshow(I2)
```



(*a*)



(*b*)

图 11.13　区域填充效果

(*a*) 原图像；(*b*) 区域填充后

## 【习　题】

1. 检测图 11.14 所示图像中的细胞结构。



图 11.14　细胞图像

2. 利用自定义滤波方法创建一种背景淡出淡入的图像效果。

# 第十二章 其他图像处理技术

**本章要点：**

★ 小波分析

★ 小波分析在图像处理中的应用

★ 分形几何

★ 神经网络

## 12.1 小波分析

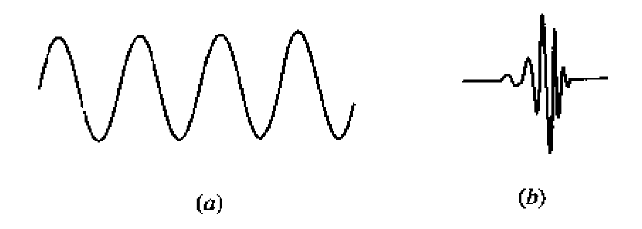
### 12.1.1 小波分析概述

从 1807 年 J. Fourier 提出傅立叶分析至今，傅立叶分析一直是信号处理的主要工具。但令人遗憾的是，在分析突变信号和非平稳信号时，傅立叶分析显得无能为力，于是寻找新的正交展开系，使之能适应突变信号和非平稳信号就成为一个新的研究热点。

小波变换正是在这一背景下产生的。小波(Wavelet)又称为子波，是一个有限的、均值为零的振荡波形。小波分析的雏形形成于 20 世纪 50 年代初的纯数学领域，但此后近 30 年里一直没有引起人们的注意。1984 年法国地质学家 J. Morlet 在分析地质数据时首先引进并使用了小波(Wavelet)这一术语。后来，数学家 Y. Meyer 将 Morlet 与早期的工作联系起来，才形成了小波分析的理论体系，使其得到了逐步的发展和应用。

小波分析与傅立叶分析有着惊人的相似之处，其基本的数学思想都来源于经典的调和分析，特别是 20 世纪 30 年代的 Little-Palay 理论。与傅立叶分析相比，小波变换是时间和频率的局域变换，能更加有效地提取信号和分析局部信号。类似于傅立叶分析，在小波分析中也有两个重要的数学实体——积分小波变换和小波级数。积分小波变换是基小波的某个函数的反射膨胀卷积，而小波级数是称为小波基的一个函数，用两种很简单的运算——二进制膨胀与整数平移来表示。通过这种膨胀和平移运算可以对信号进行多尺度的细致表示和动态分析，从而能够解决傅立叶变换不能解决的许多困难问题。图 12.1 给出了一个典型的基于傅立叶变换的正弦波和小波。与基于傅立叶变换的正弦波相比，小波不再是从极小值一直扩展到无穷大，而且也不再是光滑可测的，它呈现出一种非周期性和不对称性。

由于小波技术可以将信号或图像分层次按小波基展开，因而可以根据图像信号的性质以及事先给定的图像处理要求确定到底要展开到哪一级为止，从而不仅能有效地控制计算



(a)　　(b)

图 12.1　正弦波与小波
(a) 正弦波；(b)小波

量，满足实时处理的需要，而且可以方便地实现通常由子带编码技术实现的累进传输编码（即采取逐步浮现的方式传送多媒体图像）。同时，小波变换具有放大、缩小和平移的数学显微镜的功能，能够很方便地产生各种分辨率的图像，从而适用于不同分辨率图像的 I/O 设备和不同传输速率的通信系统。显然，这种处理方式为图像的并行处理提供了理论依据。

由于小波变换分析具有以上这些优点，所以在最近颁布的运动图像压缩标准 MPEG 4 中的视觉纹理模式就支持视觉纹理和静态图像编码，这种模式基于零高度树小波算法，在非常宽的比特率范围内具有很高的编码效率。除了具有很高的压缩效率之外，它还提供了空间和质量的可缩放性，以及对任意形状目标的编码。其空间可缩放性高达 11 级，质量的可缩放性具有连续性。

当前小波研究的一个迫切问题是如何将小波研究所取得的重要成果转化为工程技术人员所掌握的重要工具，使之尽快应用到工程技术实践中去，特别是将小波分析很好地用于多媒体图像和信号处理中。这些年来关于小波变换图像压缩算法的研究和应用都十分活跃。国外一些公司将这种技术用于 Internet 环境的图像数据传输中，以提供商业化的服务，这对于缓解网络带宽不足、加快图像信息传播速度起到了很好的推进作用。

综上所述，由于小波分析克服了傅立叶分析的许多弱点，因此它不仅可以用于图像压缩，还可以用于许多其他领域，如信号分析、静态图像识别、计算机视觉、声音压缩与合成、视频图像分析、CT 成像、地震勘探和分形力学等领域。总之，可以说凡能用傅立叶分析的地方都可以进行小波分析，其应用前景十分广阔。

### 12.1.2　连续小波变换

假设 $\varphi(t)\in L^2(R)$，定义：

$$\Psi(\omega)=\int_{-\infty}^{\infty}\Psi(t)\ \mathrm{e}^{-\mathrm{j}\omega t}\,\mathrm{d}t \tag{12.1}$$

如果 $\Psi(\omega)$满足以下条件：

$$C_{\Psi}=\int_{-\infty}^{\infty}\frac{|\Psi(\omega)|^2}{|\omega|}\,\mathrm{d}\omega<\infty \tag{12.2}$$

那么下式就称为小波函数：

$$\Psi_{a,b}=\frac{1}{\sqrt{|a|}}\Psi\left(\frac{x-b}{a}\right)\quad a\neq 0\ 且\ a\in R,\ b\in R \tag{12.3}$$

其中，$\Psi(t)$称为基小波函数，$a$ 为尺度因子，$b$ 为位移因子。从式(12.3)可以看出，小波函数就是一个满足公式(12.2)的函数经过伸缩和平移而得到的一族函数。

我们知道，傅立叶分析的过程可以用傅立叶变换进行数学描述

$$F(\omega)=\int_{-\infty}^{\infty}f(t)\,\mathrm{e}^{-\mathrm{j}\omega t}\,\mathrm{d}t \tag{12.4}$$

事实上傅立叶变换就是所有时间上的信号与复指数的乘积之和(复指数可以分解为实部和虚部正弦成分)，傅立叶变换的结果就是傅立叶系数 $F(\omega)$。同样的，连续小波变换(CWT)定义为所有时间上的信号 $f(x)$ 与小波函数 $\Psi$ 的乘积之和：

$$W_f(a,b)\leqslant f(x)$$

$$\Psi_{a,b}\geqslant\int_{-\infty}^{\infty}f(x)\Psi_{a,b}(x)\,\mathrm{d}x \tag{12.5}$$

其逆变换为

$$f(x)=\frac{1}{C_\Psi}\int_0^{\infty}\int_{-\infty}^{\infty}W_f(a,b)\Psi_{a,b}(x)\,\mathrm{d}b\,\frac{\mathrm{d}a}{a^2} \tag{12.6}$$

从式(12.4)和(12.5)可以看出，一个一维函数的连续小波是一个双变量函数，因此称连续小波变换是超完备的，因为它要求的存储量和代表的信息量都显著增加了。如果 $f(x,y)$ 是一个二维函数，那么它的连续小波变换定义如下：

$$W_f(a,b_x,b_y)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}f(x,y)\Psi_{a,b_x,b_y}(x,y)\,\mathrm{d}x\,\mathrm{d}y \tag{12.7}$$

其中，$b_x$ 和 $b_y$ 表示在两个维度上的平移。二维连续小波逆变换的定义如下：

$$f(x,y)=\frac{1}{C_\Psi}\int_0^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}W_f(a,b_x,b_y)\Psi_{a,b_x,b_y}(x,y)\,\mathrm{d}b_x\,\mathrm{d}b_y\,\frac{\mathrm{d}a}{a^3} \tag{12.8}$$

从小波变换的定义可以看出，小波变换的基本思想与傅立叶变换类似，就是用信号在由一族基函数张成的空间的投影中来表征该信号。但是，这一族函数具有一个显著的特点，即该函数系是通过一个基本小波函数的不同尺度的伸缩和平移构成的，其时宽与带宽的乘积很小，而且在时间和空间上很集中。

与信号处理中常用的短时傅立叶分析相比，小波变换的分辨力单元随尺度因子而变化，当 $a$ 增大时，频率分辨力提高，但时间分辨力降低；当 C 变小时，时间分辨力提高，但频率分辨力降低。所以当分析信号的突变时刻时，可以选择小的 $a$，使时域分辨力提高。考虑一个存在不连续点的正弦信号，假设该信号的不连续情况非常细微，几乎不可见(参见图 12.2)，这样的信号在实际中经常出现，如电源波动或噪声开关信号等。

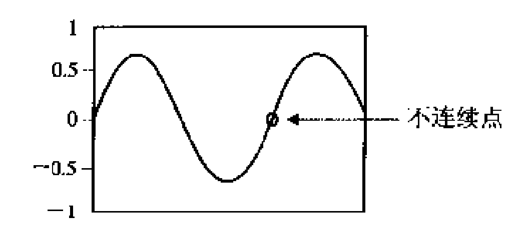



图 12.2　存在细微不连续点的信号

对于这种信号的傅立叶系数(如图 12.3($a$)所示)不能够清楚地显示出不连续点的位置，但是其小波系数却能够非常清晰地说明在时间上不连续点的准确位置(如图 12.3($b$)所示)。

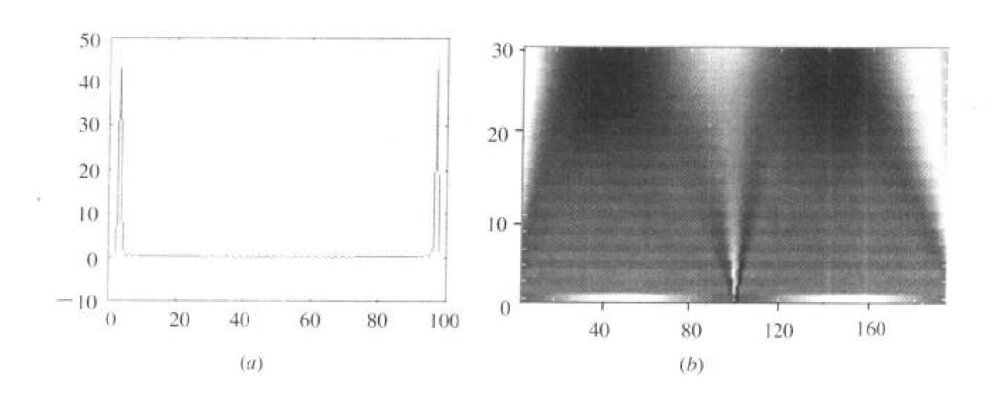



图 12.3　傅立叶变换系数与小波变换系数比较
(a) 傅立叶变换系数；(b) 小波变换系数

### 12.1.3　离散小波变换

显然，如果要计算每一个尺度上的小波变换系数，那将是一项非常庞大的任务，并且得到的数据量也是非常巨大的。为此，在实际应用中通常都是选择某个尺度集进行小波变换系数计算的。采用 2 的幂次作为尺度得到的结果通常都是非常有效而且符合精度要求的，我们把建立在这一基础上的小波变换称为离散小波变换(DWT)。

1988 年 Mallat 提出了一种利用滤波器实现 DWT 的算法，该算法就是信号处理中非常著名的双通道子带编码方法。双通道子带编码主要涉及三项技术：滤波器族理论、尺度分析和子带编码，该算法使用的滤波器基本结构如图 12.4 所示。

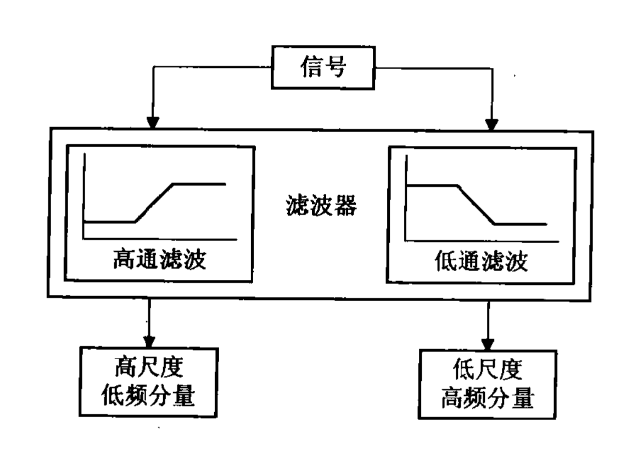



图 12.4　双通道子带编码滤波器的基本结构

虽然傅立叶变换可以反映信号的整个内涵，但是表现形式却不够直观，而且噪声会导致信号频谱复杂化。在信号处理领域一直都是使用一族带通滤波器将信号分解为不同频率分量的，即将输入信号 $f(x)$并行送到带通滤波器族 $H_i(x)$中，假设每一个滤波器相应的输出为 $g_i(x)$，则

$$g_i(x) = \int_{-\infty}^{\infty} f(t)h_i(x-t)\,\mathrm{d}t \tag{12.9}$$

构造一族滤波器 $H_i(x)$时要满足以下条件：

$$\sum_{i=1}^{\infty} H_i(s) = 1 \quad \Rightarrow \quad \sum_{i=1}^{\infty} g_i(x) = f(x) \tag{12.10}$$

这里 $g_i(x)$就是一组小波变换系数，而$\{h_i(x)\}$就是一个小波函数集。

滤波器族理论提供了一个振荡信号分析的方便手段，但是在图像分析中，我们关心的并不是真正的振荡信号，而是信号的一个或一部分周期。为了更仔细地观察这部分信息，

使用不同的小波变换尺度来膨胀或收缩小波基，从而实现信号的多分辨率。

子带编码的目的在于将信号分解为带通滤波分量，并用一种无冗余、无重建误差的方式来表示这些分量。选择一个半带低通滤波器，其通频带为$[-s_{N/2}, s_{N/2}]$，是整个频带的低频半带。可以证明，半带低通滤波后的信号可以使用二倍间隔时间来采样而不丢失信息，在重建时可以采用增频技术进行信号无误差重建。对于信号进行半带高通滤波同样具有这样的性质，因而通过对低半带和高半带信号进行编码就能够进行无冗余、无重建误差的信号采样。

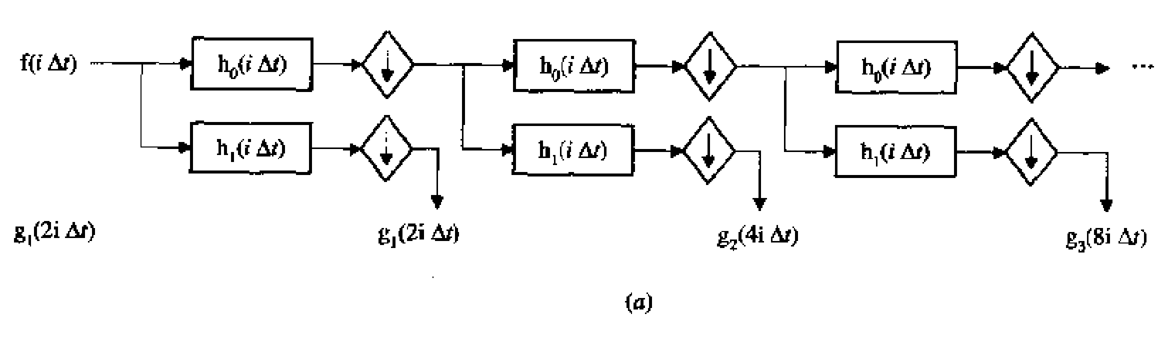
双通道子带编码就是利用以上三种技术来实现信号的快速DWT的。算法使用的基本小波为

$$h(t) = \delta(t) - \mathrm{sinc}(at)$$

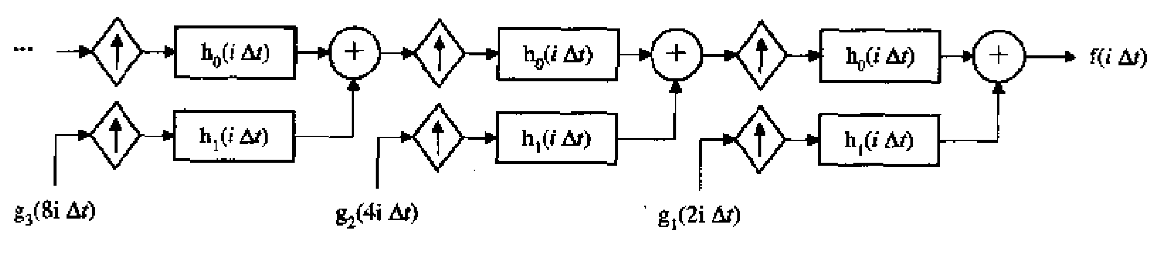
小波函数为

$$2^{-j/2}h(2^j t - n)$$

每次迭代中小波函数$\{h_i(x)\}$的尺度都将增大一倍。假设信号采样数为$N$，用该算法对低半带输出信号再进行一次半带子带编码，因而获得$N/2$个高半带采样和两个$N/4$点的子带采样，它们分别对应于区间$[0, s_N]$的第一、二个1/4区域。重复进行这一过程，在每一步都保留高半带信号，进行低半带信号的编码，直到得到了一个只有一个点的低半带信号为止，最终的小波变换系数就是这个低半带点再加上用子带编码的高半带信号的和。另外，信号通过滤波器后将生成与原始信号相同采样数目的高尺度低频分量和低尺度高频分量。假设数字信号有1000个采样，那么就会得到2000个分量采样结果。虽然这2000个采样都是有用的，但是由于其数据量过大，所以实际中常常对两个滤波器得出的信号再利用欠采样技术进行DWT计算。整个过程如图12.5(*a*)所示，图中的下箭头框图表示欠采样过程，最前面的$N/2$个系数来自于信号的高半带，接下来的$N/4$个点来自于第二个四分之一带，以此类推。如果按照以上过程的逆过程进行计算就可以得到逆小波变换，如图12.5(*b*)所




(*a*)




(*b*)

图 12.5　离散小波变换与逆变换算法

(*a*) 快速DWT；(*b*) 快速逆DWT

示，图中向上的箭头表示增频技术。

下面使用一个例子说明快速 DWT 算法的 MATLAB 实现过程。调用以下 MATLAB 代码来生成如图 12.6(*a*)所示的信号：

```
load leleccum;
s = leleccum(1:3920);
subplot(1,3,1),plot(s)
```

调用 MATLAB 小波工具箱中的 dwt 函数来实现快速 DWT 算法，调用格式如下：

```
[ca cd]=dwt(x,'wname')
```

其中，x 是要进行小波变换的向量，wname 是小波名称，ca 和 cd 分别是变换所得的高尺度低频分量和低尺度高频分量。如图 12.6(*a*)、12.6(*b*)所示。接着调用以下代码对信号 s 进行变换，图 12.6(*a*)所示信号的高尺度低频分量和低尺度高频分量分别如图 12.6(*b*)和 12.6(*c*)所示。

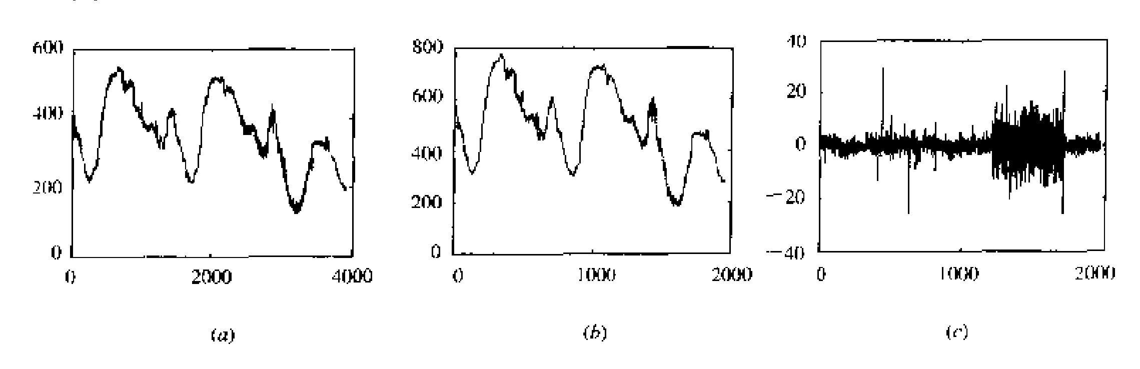



图 12.6　信号及其小波变换结果

(*a*) 原始信号；(*b*) 高尺度低频分量；(*c*) 低尺度高频分量

```
[ca1 cd1]=dwt(s,'db1');
subplot(1,3,2),plot(ca1)
subplot(1,3,3),plot(cd1)
```

调用函数 upcoef 根据高尺度分量和低尺度分量进行重构：

```
ls=length(s);
a1 = upcoef('a',ca1,'db1',1,ls);    %调用 upcoef 函数进行一维系数直接重构
d1 = upcoef('d',cd1,'db1',1,ls);
figure,plot(a1+d1)
```

重构结果如图 12.7 所示。从图中可以看出，重构是没有误差的。

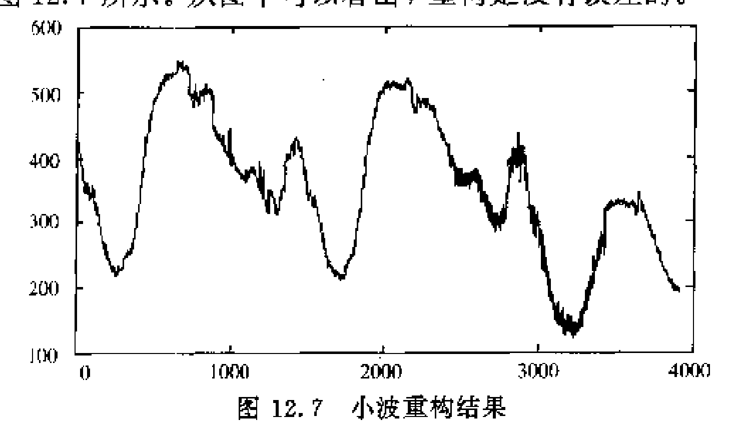



图 12.7　小波重构结果

## 12.2　小波分析在图像处理中的应用

小波分析的应用领域十分广泛，其中包括数学分析、信号分析、图像处理、量子力学、理论物理、军事电子对抗与武器的智能化、计算机分类与识别、音乐与语言的人工合成、医学成像与诊断、地震勘探数据处理和大型机械的故障诊断等方面。例如，在数学方面，它已用于数值分析、构造快速数值方法、曲线曲面构造、微分方程求解、控制论等领域；在信号分析方面，它主要用于滤波、去噪声、压缩、传递等领域；在图像处理方面，它已应用于图像的压缩、分类、识别与诊断等领域；在医学成像方面，它已应用于减少 B 超、CT、核磁共振成像的时间，以及提高图像分辨率等领域。

在图像处理方面，小波变换可以实现诸如边界扭曲、图像去噪声和图像压缩等功能。下面通过几个例子说明小波变换在这些方面的应用。以下例子中涉及到的术语请读者参考有关小波变换专著来加以学习。

**例 12.1**　利用小波变换进行边界扭曲。

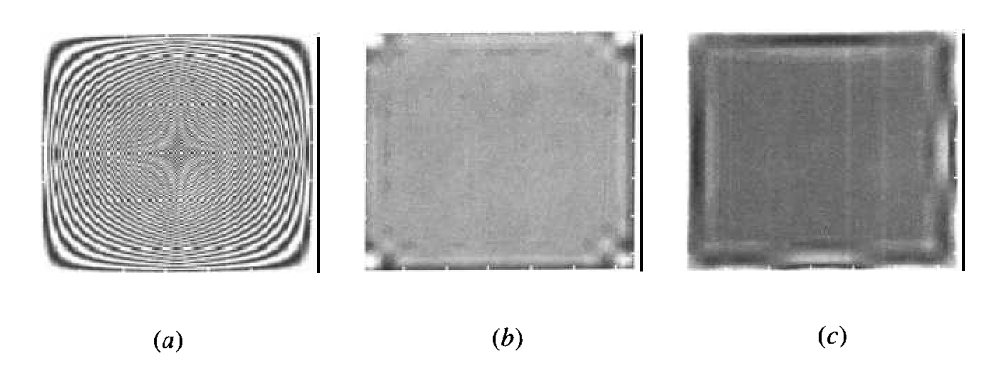
首先装载如图 12.8(*a*)所示的典型图像，设置 DWT 的填充模式为零填充：

```
load geometry;        %将图像装载到工作平台中，可以通过变量 X 访问该图像
subplot(2,2,1),image(X)
dwtmode('zpd')
```

接着利用 sym4 小波对图像进行变换，然后进行单支重构：

```
lev = 3; [c,s] = wavedec2(X,lev,'sym4');    %调用 wavedec2 函数进行多级小波分解
a1= wrcoef2('a',c,s,'sym4',lev);    %调用 wrcoef 函数根据一维变换系数进行单支重构
subplot(2,2,2),image(a1);
```

变换结果如图 12.8(*b*)所示。为了充分说明小波变换对图像边界的扭曲作用，下面采用另外一种边界填充技术——光滑填充方法，然后使用同样的小波变换对填充图像进行变换，变换结果如图 12.8(*c*)所示。



(*a*)　　(*b*)　　(*c*)

图 12.8　边界扭曲效果

(*a*) 原图像；(*b*) 经 sym4 小波变换、三阶重构后；(*c*) 经光滑填充、小波变换后

```
dwtmode('spd')
[c,s] = wavedec2(X,lev,'sym4');
a3 = wrcoef2('a',c,s,'sym4',lev);
subplot(2,2,4),image(a3);
```

**例 12.2** 利用小波变换消除图像噪声。

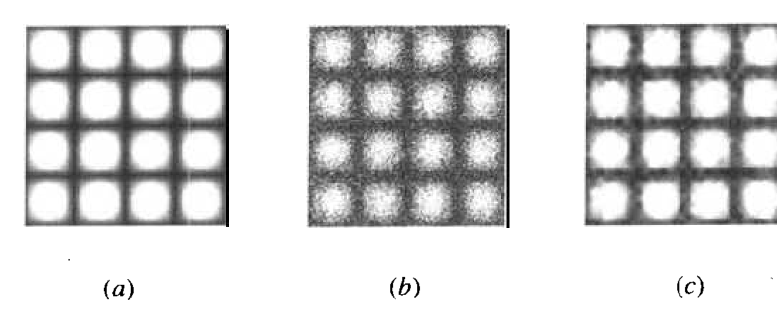
以图 12.9(*a*)所示图像为例，装载该图像并给该图像添加随机噪声：

```
load sinsin
suplot(1,3,1),image(X)
init = 2055615866; randn('seed',init);
x = X + 18 * randn(size(X));
suplot(1,3,2),image(x)
```

噪声图像如图 12.9(*b*)所示，调用 wdencmp 对该图像进行去噪声：

```
[thr,sorh,keepapp] = ddencmp('den','wv',x);
                    %调用 ddencmp 函数求取去噪声缺省值
xd = wdencmp('gbl',x,'sym4',2,thr,sorh,keepapp);
suplot(1,3,3),image(xd)
```

去噪声结果如图 12.9(*c*)所示。总体来说，小波变换的去噪声结果还是令人满意的。



(*a*) (*b*) (*c*)

图 12.9 去噪声效果

(*a*) 原图像；(*b*) 填加随机噪声后；(*c*) 去噪声后

**例 12.3** 利用小波变换进行图像压缩。

首先将图 12.10(*a*)所示图像装载到工作平台中：

```
load woman;
x = woman(100:200,100:200);
suplot(1,3,1),image(x)
```

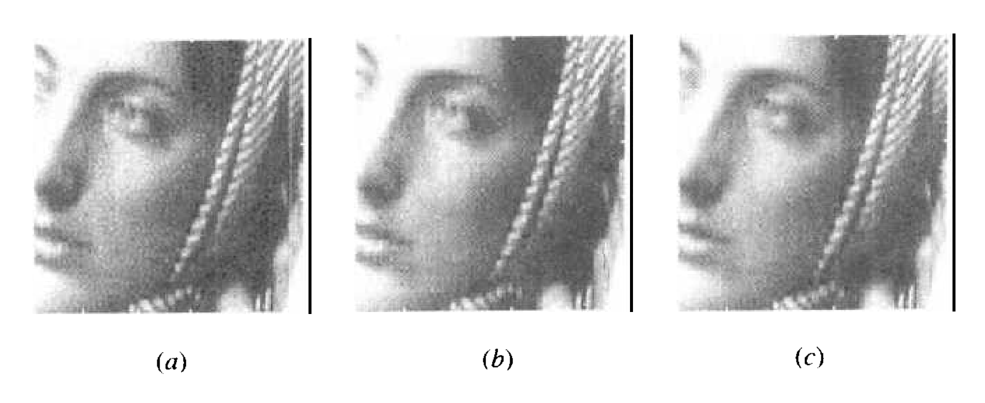
使用全局阈值对图像进行小波变换压缩，并对压缩后的图像进行二维重构：

```
n = 5; w = 'sym2';
[c,l] = wavedec2(x,n,w);
thr = 20;          %小波变换阈值
[xd,cxd,lxd,perf0,perfl2] =wdencmp('gbl',c,l,w,n,thr,'h',1);
suplot(1,3,2),image(xd)
```

重构结果如图 12.10(*b*)所示。也可以对图像分别进行三个方向上的阈值小波变换。这种变换压缩方式下的图像二维重构结果如图 12.10(*c*)所示。比较两种方法可知，后者的压缩效果要好一些。

```
thr_h = [17 18];      % 水平阈值
thr_d = [19 20];      % 对角阈值
thr_v = [21 22];      % 垂直阈值
thr = [thr_h ; thr_d ; thr_v]
```

```
[xd,cxd,lxd,perf0,perfl2] = wdencmp('lvd',x,'sym8',2,thr,'h');
suplot(1,3,3),image(xd)
```



(a)　　(b)　　(c)

图 12.10　图像压缩效果

(a) 原图像；(b) 经阈值变换、重构后；(c) 经三个方向阈值变换、重构后

## 12.3　分形几何

### 12.3.1　分形几何概述

在现实世界中，几何形体可以分为两大类：一类是规则的、光滑的，可以用直线段、平面片或小六面体来逼近的形体，研究这一范畴的学科称为传统几何学；另一类则是不光滑的、不规则的，具有精细的结构或自相似特征，不能用传统的几何语言描述的自然形态，如海岸线、山形、河川、树木、闪电等自然景物，以及微观世界中复杂且精细的结构、宏观世界里天体演变的各种形态，我们称之为分形，研究这一范畴的学科称为分形几何学。

分形几何学是由 Mandelbrot 首先提出并发展为系统理论的。Mandelbrot 在研究英国海岸线的复杂边界时发现，在不同比例的地图上会测出不同的海岸线长度，这正是欧几里德几何无法解释的。在研究中，他将测量长度与放大比例(尺度)分别取对数，所对应的二维坐标点存在一种线性关系，此线性关系可用一个定量参数(称该参数为分维数)来描述。由此，Mandelbrot 将其进一步发展成为分形几何理论，该理论可以产生许多分形集图形和曲线，如 Mandelbrot 集、Cantor 集、Koch 曲线、Sierpinski 地毯等，还可描述复杂对象的几何特性。

与欧氏几何比较，分形几何主要有以下特点：

■ 描述对象虽然很复杂、不规则，但不同尺度上是具有规则性或相似性的；

■ 欧氏几何具有有限标度，而理想分形具有无限的几何标度，而无特征长度；

■ 欧氏几何用整数维描述形体特征，而具有分形的复杂曲线，其分维数是大于 1 的非整数，具有分形的表面分维则是大于 2 的非整数。

分形理论为描述事物的复杂性提供了有力的数学模型、定量参数和方法。目前分形几何已渗透到数学、物理、化学、天文，以及心理、社会学等许多应用领域。分形学使人们对自然界和人类社会的认识提高到一个崭新的高度。分形理论与计算机技术结合后更是发展迅猛，目前已经成为一门跨学科、非线性并且相当活跃的学科，其理论研究和应用已经深

入到人类活动的方方面面，并取得了令人瞩目的成果。

随着分形图像压缩技术的不断改进和完善，它在图像压缩领域中将越来越显示出优势。1988 年 Barnsley 采用迭代函数系统 IFS 和递归迭代函数系统 RIFS 方法对几幅图像进行压缩编码，获得了高达 10000：1 的压缩比。另外，用分形方法还能在计算机上模拟自然景物、动画制作和建筑物配景等，在影视制作中能生成奇峰异谷等独特场景，产生新奇、美丽的景色。

由于分形几何是一门较新的学科，目前还处在不断的发展过程中，所以 MATLAB 软件没有提供有关分形几何的应用函数，感兴趣的读者可以自己编写有关函数，将其添加到 MATLAB 工具箱中进行应用，这里仅对其概念和应用方法作简单介绍。

### 12.3.2 分形理论基本概念

Pentland 通过对自然景物纹理图像的研究，证明大多数自然景物的灰度图像是满足多向同性分数布朗运动(FBM)场模型的，因此对 FBR 场的参数研究，可以有效地分析分形图像。

FBM 是一种典型的分形模型，可以很好地描述分形信号：它是连续不可导的一种非平稳随机过程，对尺度变化具有相似性。FBM 的增量是平稳的、零均值 Gaussion 随机过程。FBM 定义如下：

设 $B_H(t)$为一随机场，对于 $c<H<1$，若满足：

$$P_r\left(\frac{B_H(t+\Delta t)-B_H(t)}{\| t \|^H}<y\right)=F(y) \tag{12.11}$$

则称 $B_H(t)$为 FBR 场，$H$ 为 Hust 系数。式(12.11)中 $P_r$ 为概率测度，$F(y)$为高斯分布函数。(12.11)式的数学期望为

$$E\{B_H(t+\Delta t)-B_H(t)\}=E\left\{y\exp\left[\frac{1}{2\pi^{1/2}}\sigma\|\Delta t\|^{2H}\right]\right\} \tag{12.12}$$

$H$ 参数的估计有时域和频域两种方法，都是基于(12.12)式推导求得的，增量标准差 6 也可以由(12.12)式得出。理想分形都是满足(12.11)式的，故称其具有无限标度，对于实际图像，由于量化效应和模型的差异，只有一段尺度空间($\varepsilon_{min}$，$\varepsilon_{max}$)使(12.11)式满足线性关系，称该尺度区间为无标度区。实际图像越接近理想分形，则无标度区间越大，即 $\varepsilon_{min}/\varepsilon_{max}$ 的值越大。在无标度区中可用线性回归方法估计 $H$ 值。

### 12.3.3 分维数估计方法

分维数 $D_F$ 可由下式通过 Hust 系数得到：

$$D_F=D+1-H \tag{12.13}$$

其中，$D$ 是拓扑维，对于曲线，$D=1$；对于 FBR 表面，$D=2$。$D_F$ 是描述分形的主要参数，一般地，当不规则曲线的 $D_F>1$，或纹理表面的 $D_F>2$ 时，认为它们具有分形性质。分维值的估计方法很多，从速度和精度考虑，比较实用的有以下几种：

■ 数盒子法：对于分形曲线，用可变尺度 $\epsilon$ 沿曲线度量长度需 $N$ 次，$N(\epsilon)$是随 $\epsilon$ 而变的，由(12.11)式可推出：

$$D=\lim_{\epsilon\to 0}\left(\frac{\ln N}{\ln\epsilon}\right) \tag{12.14}$$

为求 $N(\varepsilon)$，在计算时，首先以不同尺寸的网状栅格覆于曲线上（$\varepsilon$ 为栅格大小），然后计算求得与曲线相交的格子数，即为 $N(\varepsilon)$，最后，利用双对数曲线估计 $H$ 值及分维值 $D_F$。同理，对于分形纹理曲面，它被包容在三维空间中，因此用小立方体代替网状栅格，同样取不同尺寸的立方体覆盖于曲面上，可得到与尺寸 $\varepsilon$ 对应的小立方体总数 $N(\varepsilon)$，进而求得分形表面的分维值。

■ 功率谱法：对图像先作傅立叶变换，使其成为频谱图，其功率谱为 $P(u)^2$，而频率半径为 $R=U^2+V^2$，作出功率谱与频率半径的双对数图，根据线性回归直线求取分维数。

■ 地毯覆盖方法：设分形表面为 $g(i,j)$，形象地用厚度为 $2\varepsilon$ 的"地毯"覆盖，则毯的上表面点集为 $t_\varepsilon(i,j)$，下表面点集为 $b_\varepsilon(i,j)$，初始状态为 $t_0(i,j)=b_0(i,j)=g(i,j)$。当厚度 $\varepsilon=1,2,3,\cdots$变化时：

$$t_\varepsilon(i,j)=\max\{t_{\varepsilon-1}(i,j)+1,\max(m,n),st_{\varepsilon-1}(m,n)\}$$
$$b_\varepsilon(i,j)=\max\{b_{\varepsilon-1}(i,j)-1,\max(m,n),sb_{\varepsilon-1}(m,n)\} \tag{12.15}$$

其中 $i=1,2,\cdots,m$；$j=1,2,\cdots,n$。$s$ 为点 $(i,j)$ 的邻域点集，则在尺度 $\varepsilon$ 下，毯的面积为

$$A(\varepsilon)=\sum_{i=1}^{m}\sum_{j=1}^{n}\frac{t_\varepsilon(i,j)-b_\varepsilon(i,j)}{2\varepsilon} \tag{12.16}$$

由 $A(\varepsilon)$ 和 $\varepsilon$ 的双对数图即可估计出分形表面的分维值。

### 12.3.4 分形方法在图像处理中的应用

分形理论在图像处理中主要用于以下几个方面：模拟自然景物生成、图像分割、纹理特征提取、纹理分类和图像编码压缩。

理想的分形图像是利用分形的自相似性，通过递归迭代方法生成的；对自然物体（如云彩、海浪、树木等图像）的分形生成要更加复杂，需多级嵌套、多级结构。一种好的分形模型可描述事物的形态、结构以及功能的多种变化。

根据离散布郎随机场的理论，若图像表面统计特性满足各向同性时，可由随机场参数 $H$ 得出表面分维数。但对于不同区域的交界处，破坏了随机场的一致性，$H$ 值就会发生奇异。利用这一类可作检测边缘、分割区域的依据，在图像上定义移动窗口，用窗内像素估计该窗口中心点的 $H$ 值（或分维），于是用估计出的所有像素的分形参数形成一幅新图，进而可提取边缘。

形状分析和纹理分析是目前分形理论应用最多的一个领域。具有分形特性的对象往往表现在边界很不规则、很复杂，用传统的周长、面积作近似描述很不适合，而采用分形参数（如分维数）及其导出形状特征可以精确地进行定量描述，为形状分析及目标识别等提供了很简洁的特征。纹理图像也具有复杂性、自相似性等特点，尤其是自然纹理很适于用分形模型来描述。纹理分割和识别主要依据纹理特征，大体分为基于统计特征和基于结构特征两种方法，主要有共生矩阵特征、功率谱特征、Law's 纹理能量函数、Marlkov 纹理随机场等。Philippe 等人对多种纹理特征进行了多种实验比较，认为分形维特征是一种非常有效的纹理特征，无论从计算精度、速度还是从判别正确率方面都比较高。

迭代函数系统（IFS）是计算机图形学、仿射几何学和分形理论的结合。IFS 可提供自然景物的分形图像构造方法用以产生各种形态的景物，还可把一幅复杂的图像通过仿射变换简化为很少的特征量，用于编码存储和传输，并可以通过拼粘原理再现原图，这使得 IFS

具有很强的数据压缩能力。

分形图像编码原理是这样的：对现实世界中的图像集合引入 Hausdorff 度量，使其形成一个完整的度量空间，它的每一个点既表示一幅图像，又是欧氏空间的一个紧子集。分形图像压缩的理论基础是迭代函数系统 IFS 定理、收缩映射定理和拼贴定理。一个迭代函数系统由一个完备的度量空间和其上的一组收缩映射组成。收缩映射定理告诉我们，函数空间中的每一个收敛映射都有一个固定点，使函数空间中的每一个点经过这个收缩映射的连续作用后，形成的点列收敛于这个固定点；迭代函数系统定理指出，每个迭代函数系统都可以构成函数空间中的一个收缩映射。于是，我们得到结论：每个迭代函数系统都决定一幅图像；而拼贴定理告诉我们：给定一幅图像 I，可以选择 $N$ 个收缩映射，这幅图像经过 $N$ 个变换得到 $N$ 个像集，每个像集都是一块小图像。如果这 $N$ 个小图像拼贴起来的图像与图像 I 之间的距离任意小，则这 $N$ 个收缩映射构成的迭代函数系统所决定的图像就任意地接近图像 I。分形图像编码的过程就是依据拼贴定理，通过给定的图像，寻找一组收缩映射，使其组成的迭代函数系统的吸引子逼近给定图像，然后记录下相应参数。相应的解码过程则是由相应参数确定迭代函数系统，并根据迭代函数系统定理，经过迭代生成图像。

分形图像压缩编码有两种基本方法。一种是人工干预的交互式分形图像编码方法。它主要是针对给定图像的形状，采用边缘检测、频谱分析、纹理分析、分维方法等传统的图像处理技术进行图像分割的。它要求被分开的每部分都有比较直观的自相似特征，然后通过寻找迭代函数系统来确定各个变换系统，再由图像中灰度分布求得各个变换的伴随概率。解码过程是采用随机迭代法来生成近似图像。这种方法的压缩比一般是相当高的。另一种方法是自适应块状分形编码方法。首先将图像分割成若干不重叠的值域块 $R_i$ 和可以重叠的定义域块 $D_j$，接着对每个 $R_i$ 寻找某个 $D_j$，使 $D_j$ 经过某个指定的变换映射到 $R_i$，以达到规定的最小误差，记录下确定 $R_i$ 和 $D_j$ 的参数及变换 $W_i$，得到一个迭代函数系统。最后对这些参数进行编码，编码过程包括对图像的分割、搜索最佳匹配、记录相关的系数等三个步骤。自适应块状分形解码方法是由编码传来的参数确定迭代函数系统，经过有限次迭代，图像会稳定下来，它趋近迭代函数系统的吸引子，该吸引子就是被编码图像的解码图。上述编码过程中搜索匹配的计算量很大，耗用时间太长，限制了分形编码的实际应用。因此，人们不断地进行研究，提出了许多改进方案，常用的改进方法有：基本四叉树分割法、基于 HV 分割法、快速覆盖式分形压缩方法和四叉树重组 QR 算法等。

## 12.4 神经网络

### 12.4.1 神经网络概述

人工神经网络(ANNs)也称为神经网络或连接模型，是对人脑或自然神经网络若干基本特性的抽象和模拟。人工神经网络是在现代神经科学的基础上提出来的，它虽然反映了人脑功能的基本特征，但远不是自然神经网络的逼真描写，而只是它的某种简化抽象和模拟。人工神经网络的研究内容相当广泛，反映了多学科交叉技术领域的特点。神经网络研究热潮的兴起是 20 世纪末人类科学技术发展全面飞跃的一个组成部分，与多种科学领域

的发展密切相关，迄今为止，在人工神经网络研究领域中，有代表性的网络模型已达数十种，而学习算法的类型则更难以统计其数量。

人工神经网络的以下几个突出的优点是使它近年来受到极大关注的主要原因：

■ 可以充分逼近任意复杂的非线性关系；

■ 所有定量或定性的信息都等势分布储存于网络内的各神经元中，故有很强的鲁棒性和容错性；

■ 采用并行分布处理方法，使得快速进行大量运算成为可能；

■ 可学习和自适应不知道或不确定的系统；

■ 能够同时处理定量、定性知识；

■ 具有联想和存储功能；

■ 具有高速寻找优化解的能力。

神经网络的研究已有较长的历史，最早的研究是20世纪40年代由心理学家Mcculloch和数学家Pitts合作提出的兴奋与抑制型神经元模型以及由Hebb提出的神经元连接强度的修改规则，他们的研究结果至今仍是许多神经网络模型研究的基础。20世纪50年代、20世纪60年代的代表性工作是Rosenblatt的感知机和Widrow的自适应性元件Adaline。1969年，Minsky和Papert合作发表了颇有影响的《Perceptron》一书，得出了消极、悲观的论点，加上数字计算机正处于全盛时期，并在人工智能领域取得显著成就，使得20世纪70年代人工神经网络的研究处于低潮。进入20世纪80年代后，传统的Von Neumann数字计算机在模拟视、听觉的人工智能方面遇到了物理上不可逾越的障碍。与此同时，Rumelhart与Mcclelland以及Hopfield等人在神经网络领域取得了突破性进展，神经网络的热潮被再次掀起。目前在研究方法上已形成了多个流派，最富有成果的研究工作包括：多层网络BP算法、Hopfield网络模型、自适应共振理论(ART)和自组织特征映射理论等。

### 12.4.2　生物神经元模型

神经系统的基本构造是神经元(神经细胞)，它是处理人体内各部分之间相互信息传递的基本单元。据神经生物学家研究的结果表明，人的一个大脑一般有1010～1011亿个神经元。如图12.11所示，每个神经元都由一个细胞体，一个连接其他神经元的轴突和一些向外伸出的其它较短分支——树突组成。

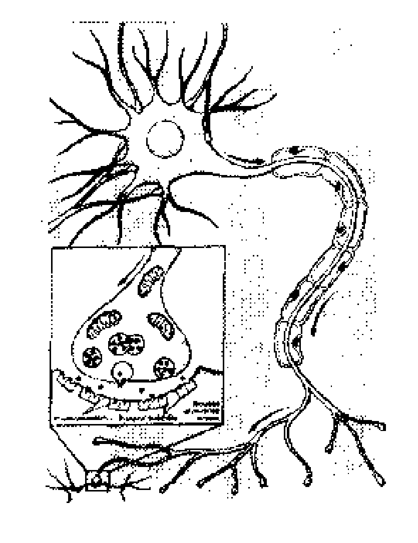


图 12.11　神经元结构图

轴突的功能是将本神经元的输出信号(兴奋)传递给别的神经元，其末端的许多神经末梢使得兴奋可以同时传送给多个神经元。神经元的树突与另外的神经元的神经末梢相连的部分称为突触。树突的功能是接受来自其它神经元的兴奋。神经元细胞体将接受到的所有信号进行简单地处理(如加权求和，即对所有的输入信号都加以考虑，且对每个信号的重视程度有所不同，这体现在权值上)后由轴突输出。

构造人工神经网络的每一个神经元模型都将模拟一个生物神经元，该神经元单元由多个输入 $x_i(i=1,2,\cdots,n)$和一个输出 $y$ 组成。中间状态由输入信号的权和表示，而输出为

$$y_j(t) = f\left(\sum_{i=1}^{n} w_{ji}x_i - \theta_j\right) \tag{12.17}$$

其中，$\theta_j$ 为神经元单元的偏置(阈值)，$w_{ji}$为连接权系数(对于激发状态，$w_{ji}$取正值，对于抑制状态，$w_{ji}$取负值)，$n$ 为输入信号数目，$y_j$ 为神经元输出，$t$ 为时间，$f$ 为输出变换函数，有时也叫作激发或激励函数。神经元模型如图 12.12 所示。

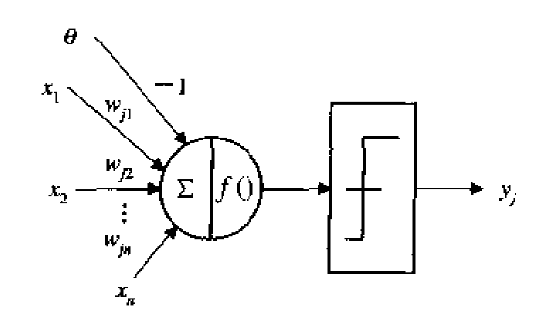



图 12.12 神经元模型

激励函数往往采用 0 和 1 二值函数或 S 形函数，常用的一种二值函数可由下式表示，其函数曲线如图 12.13(*a*)所示。

$$f(x) = \begin{cases} 1 & x \geqslant x_0 \\ 0 & x < x_0 \end{cases} \tag{12.18}$$

一种常规的 S 形函数如图 12.13(*b*)所示，可由下式表示：

$$f(x) = \frac{1}{1+\mathrm{e}^{-\theta x}} \quad 0 < f(x) < 1 \tag{12.19}$$

常用双曲正切函数(如图 12.13(*c*)所示)来取代常规 S 形函数，因为 S 形函数的输出均为正值，而双曲正切函数的输出值可为正或负。双曲正切函数如下式所示：

$$f(x) = \frac{1-\mathrm{e}^{-\theta x}}{1+\mathrm{e}^{-\theta x}} \quad -1 < f(x) < 1 \tag{12.20}$$

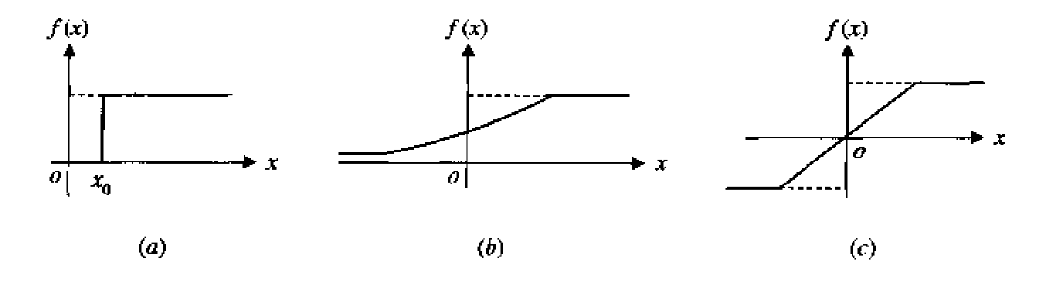



图 12.13 某些变换(激发)函数
(*a*) 激励函数；(*b*) S 形函数；(*c*) 双曲正切函数

### 12.4.3 神经网络模型及分类

神经网络是一个具有高度非线性的超大规模的连续时间动力系统，其最主要特征为连续时间非线性、网络全局作用、大规模并行分布处理以及高度的鲁棒性和学习联想能力。同时，它还具有一般非线性动力系统的共性，即不可预测性、耗散性、不可逆性、高维性、广泛联结性与自适应性等，因此它实际上是一个超大规模非线性连续时间自适应信息处理系统。

在人工神经网络系统中，信息的存储与处理是合二为一的，即信息的存储体现在神经元互连的分布上，并以大规模并行分布方式处理。网络的信息处理由神经元之间的相互作用来实现；知识与信息存储表现为网络元件互连间分布式的物理联系；网络的学习和识别决定于各种神经元连接权系的动态演化过程。这种并行处理决不是简单地以空间复杂性代替时间复杂性，而是反映了完全不同的计算原理。从数学观点看，可以把神经网络看作是由大量子系统组成的大系统，系统的最终行为完全由它的吸引子决定，如果视动力系统的稳定吸引子为记忆的话，那么从初态向吸引子流动的过程就是寻找记忆的过程。初态可以认为是给定的有关记忆的部分信息。换言之，流动的过程就是从部分信息找出全部信息的过程，这就是联想记忆的基本原理。进一步而言，若视动力系统的稳定吸引子为系统计算能量函数的极小点，系统最终会流向期望的最小点，计算也就在运动过程中悄悄地完成了。运动的时间就是计算时间，这就是神经网络计算机的基本原理。

如果按照人工神经网络对生物神经系统的不同组织层次和抽象层次的模拟来划分，神经网络模型可分为以下几种类型：

■ 神经元层次模型：它的研究工作主要集中在单个神经元的动态特性和自适应特性上，它用于探索神经元对输入信息有选择的响应和某些基本存储功能的机理；

■ 感知器模型：它由数种相互补充、相互协作的神经元组成，用于完成某些特定的任务，如模式识别、机器人控制等；

■ 网络层次模型：它是由许许多多相同神经元相互连接而形成的网络，它是从整体上研究网络的集体特性；

■ 神经系统层次模型：它一般是由多个不同性质的神经网络构成，以模拟生物神经的更复杂或更抽象的性质，如自动识别、概念形成、全局稳定性控制等；

■ 智能型模型：这是最抽象的层次，多以语言形式模拟人脑信息处理的运行、算法和策略，这些模型试图模拟类似感知、问题求解等基本的人脑思维过程。

根据各层次模型间的连接方式不同，神经网络可分成以下几种类型：

■ 不含反馈的前向网络：神经元分层排列，组成输入层、隐层和输出层。每一层的神经元只接受前一层神经元的输入。输入模式经过各层的顺次变换后，送到输出层进行输出；

■ 从输出层到输入层有反馈的前行网络：它可用来存储某种模式序列；

■ 层内有相互结合的前向网络：通过层内神经元间的相互结合，可以实现同一层内神经元之间的横向抑制或兴奋机制，这样可以限制每层内能同时动作的神经元的数目，或者把每层内的神经元分为若干组，让每一组作为一个整体来动作；

■ 相互结合型网络：这种网络是在任意两个神经元之间都可能有连接。在无反馈的前向网络中，信号一旦通过某个神经元，过程就结束了，而在相互结合的网络中，信号要在神经元之间反复往返传递，网络处在一种不断改变状态的动态过程之中。从某初态开始，经过若干次的变化，才会到达某种平衡状态，根据网络的结构和神经元的特性，还有可能进入周期性振荡或其它（如混沌等）状态。

### 12.4.4　典型神经网络简介

**1. 多层感知网络(误差逆传播神经网络)**

在 1986 年以 Rumelhart 和 McCelland 为首的科学家完整地提出了误差逆传播学习算法，并被广泛接收。多层感知网络是一种具有三层或三层以上阶层的阶层型神经网络。典型的多层感知网络是三层的、前馈的阶层网络，即输入层 I、中间层(也称隐含层)J、输出层 K，如图 12.14 所示。相邻层之间的各神经元实现全连接，即下一层的每一个神经元与上一层的每一个神经元都实现全连接，而且每层各神经元之间无连接。

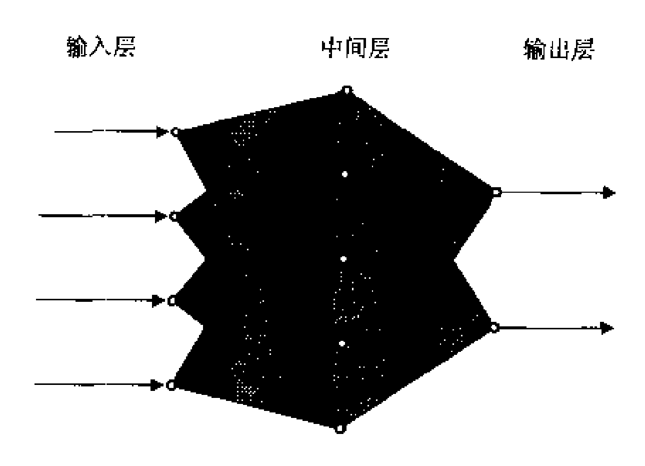



图 12.14　多层感知网络结构图

多层感知网络以一种有教师示教的方式进行学习：首先由教师对每一种输入模式设定一个期望输出值，然后对网络输入实际的学习记忆模式，并由输入层经中间层向输出层传播(称为模式顺传播)。实际输出与期望输出的差即是误差。按照误差平方最小这一规则，由输出层往中间层逐层修正连接权值，此过程称为误差逆传播，误差逆传播神经网络简称 BP 网。随着模式顺传播和误差逆传播过程的交替反复进行。网络的实际输出逐渐向各自所对应的期望输出逼近，网络对输入模式的响应的正确率也不断上升。通过此学习过程，确定下来各层间的连接权值之后就可以工作了。

由于 BP 网及误差逆传播算法具有中间隐含层，并有相应的学习规则可寻，使得它具有对非线性模式的识别能力。特别是其数学意义明确、步骤分明的学习算法，更使其具有广泛的应用前景。目前，在手写字体的识别、语音识别、文本一语言转换、图像识别以及生物医学信号处理方面已有实际的应用。虽然 BP 网具有以上优点，但是它并不十分完善，还存在以下一些主要缺陷：学习收敛速度太慢、网络的学习记忆具有不稳定性，即当给一个训练好的网提供新的学习记忆模式时，将使已有的连接权值被打乱，导致已记忆的学习模式的信息的消失。

**2. 竞争型神经网络**

竞争型神经网络是基于人的视网膜及大脑皮层对刺激的反应而引出的。神经生物学的研究结果表明：生物视网膜中有许多特定的细胞对特定的图形(输入模式)比较敏感，并使

得大脑皮层中的特定细胞产生较大的兴奋，而其相邻的神经细胞的兴奋程度被抑制。对于某一个输入模式，通过竞争在输出层中只激活一个相应的输出神经元。许多输入模式在输出层中将激活许多个神经元，从而形成一个反映输入数据的“特征图形”。

竞争型神经网络是一种以无教师方式进行网络训练的网络，它通过自身训练自动地对输入模式进行分类。竞争型神经网络及其学习规则与其它类型的神经网络和学习规则相比有其自己鲜明的特点。在网络结构上，它既不像阶层型神经网络那样各层神经元之间只有单向连接，也不像全连接型网络那样在网络结构上没有明显的层次界限，它一般是由输入层(模拟视网膜神经元)和竞争层(模拟大脑皮层神经元，也称为输出层)构成的两层网络。两层之间的各神经元实现双向全连接，而且网络中没有隐含层。有时竞争层各神经元之间还存在横向连接。竞争型神经网络的基本思想是网络竞争层各神经元竞争对输入模式的响应机会，最后仅有一个神经元成为竞争的胜者，并且只将与获胜神经元有关的各连接权值进行修正，使之朝着更有利于它竞争的方向调整。神经网络工作时，对于某一输入模式，网络中与该模式最相近的学习输入模式相对应的竞争层神经元将有最大的输出值，即用竞争层获胜神经元来表示分类结果。这是通过竞争得以实现的，实际上也就是网络回忆联想的过程。

除了竞争的方法外，还有通过抑制手段获取胜利的方法，即网络竞争层各神经元抑制所有其它神经元对输入模式的响应机会，从而使自己“脱颖而出”，成为获胜神经元。除此之外还有一种称为侧抑制的方法，即每一个神经元只抑制与自己邻近的神经元，而对远离自己的神经元不抑制，这种方法常常用于图像边缘处理中，以解决图像边缘的缺陷问题。由于竞争型神经网络仅以输出层中的单个神经元代表某一类模式，所以一旦输出层中的某个输出神经元损坏，则会导致该神经元所代表的模式信息全部丢失。

**3. Hopfield 神经网络**

1986 年美国物理学家 J. J. Hopfield 利用非线性动力学系统理论中的能量函数方法研究反馈人工神经网络的稳定性，并利用此方法建立求解优化计算问题的系统方程式，提出了基本的 Hopfield 神经网络。该网络是一个由非线性元件构成的全连接型单层反馈系统，网络中的每一个神经元都将自己的输出通过连接权传送给所有其它神经元，同时又都接收所有其它神经元传递过来的信息(参见图 12.15)。由于网络中的神经元在 $t$ 时刻的输出状态实际上都间接地与自己在 $t-1$ 时刻的输出状态有关，所以 Hopfield 神经网络是一个反馈型的网络，其状态变化可以用差分方程来表征。

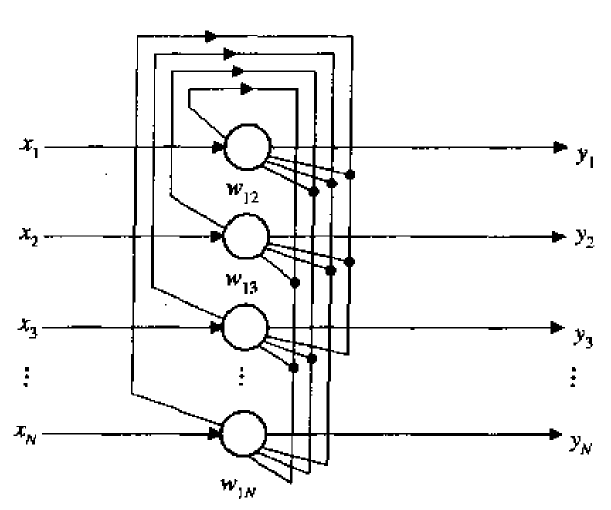


图 12.15　Hopfield 神经网络的结构

反馈型网络的一个重要特点就是它具有稳定状态。当网络达到稳定状态的时候，也就是它的能量函数达到最小的时候。注意，这里的能量函数不是物理意义上的能量函数，而是在表达形式上与物理意义上的能量概念一致，用于表征网络状态的变化趋势，并可以依据 Hopfield 工作运行规则不断进行状态变化，最终能够达到某个极小值的目标函数。网络

收敛就是指能量函数达到极小值。如果把一个最优化问题的目标函数转换成网络的能量函数，把问题的变量对应于网络的状态，那么 Hopfield 神经网络就能够用于解决优化组合问题。

Hopfield 网络通常只接受二进制输入(0 或 1)以及双极输入(+1 或 −1)，它含有一个单层神经元，每个神经元与所有其它神经元连接，形成递归结构。对于同样结构的网络，当网络参数(指连接权值和阀值)有所变化时，网络能量函数的极小点(称为网络的稳定平衡点)的个数和极小值的大小也将变化，因此，可以把所需记忆的模式设计成某个确定网络状态的一个稳定平衡点。若网络有 $M$ 个平衡点，则可以记忆 $M$ 个记忆模式。当网络从与记忆模式较靠近的某个初始状态出发后，网络按 Hopfield 运行规则进行状态更新，最后网络的状态将稳定在能量函数的极小点，这样就完成了部分信息的联想过程。

Hopfield 神经网络的能量函数是朝着梯度减小的方向变化的，但它仍然存在一个问题，那就是一旦能量函数陷入到局部极小值，它将不能自动跳出局部极小点而到达全局最小点，因而无法求得网络最优解。这一问题可以通过采用模拟退火算法或遗传算法来加以解决，在此不再一一介绍。

### 12.4.5 神经网络在图像处理中的应用

传统的图像处理有两个难以克服的缺点：一是图像数据量较大，计算机目前还难以达到足够的速度和存储容量进行图像的实时处理；二是由于图像都或多或少地存在一定的畸变和噪声，所以在进行图像处理时很有可能会导致图像处理操作的失效，因而在对图像进行处理之前都需要一定的先验知识。基于神经网络的计算机技术能够实现多机并行处理，不但能够提高计算机的处理速度，而且可以利用多处理器同时工作的优点提取更多的图像相关性信息，因而可以更好地从全局的角度把握图像的特征。由此可见，神经网络技术必定会在图像处理领域中大显身手。

神经网络技术在图像复原、边缘检测、边缘增强和图像分割等领域都有着广泛的应用。如果希望详细了解神经网络在这些图像处理操作中的应用，那么就需要对神经网络有一个较为深入的了解，因而这里仅给出一个简单的神经网络应用实例，通过这个例子读者可以初步领会到神经网络技术的基本概念和 MATLAB 实现方法。

**例 12.4**　利用神经网络技术通过对信号进行学习，从而预报下一时刻的信号。

本例中给出两个有较强相关性的信号，利用神经网络技术分析这两个信号的关系，然后进行信号预报。

首先调用以下语句创建一个信号：

```
time=1:0.01:2.5;
X=sin(sin(time).*time*10);
P=con2seq(x);
```

将信号 P 向左平移，幅值扩大两倍后再与原始信号叠加，从而得到另一个信号 T：

```
T=con2seq(2*[0 X(1:(end-1))]+X);
Plot(time,cat(2,P{:},time,cat(w,T{:}),'-')
```

创建一个线性神经网络来学习信号 P 和 T 之间的时间－位移相关性。显然，这个网络一定要稍有延迟，这样才能进行相关性学习。

```
Net=newlin([-3 3],1,[0 1],0.1);
```

在以上语句中，[-3 3]是预期的输入范围，newlin 函数的第二个参数是每一层的神经元数目。[0 1]指定一个无延迟输入和一个有延迟输入。最后一个参数表示学习速率。下面就可以调用 adapt 函数来学习这个信号了：

```
[net,Y,E,Pf]=adapt(net,P,T);
```

用实线表示信号 T，点横线表示学习过程 Y，星号表示学习误差，绘制这个网络的学习过程(如图 12.16 所示)：

```
plot(time,cat(2,Y{:}),time,cat(2,T{:}),'-*',time,cat(2,E{:}),'-',...
    [1 2.5],[0 0],'k');
```

从图 12.16 中可以看出，大约两秒钟后这个网络就学会了信号 T 的产生方式，这之后进行预报的结果与实际信号的误差几乎为零。

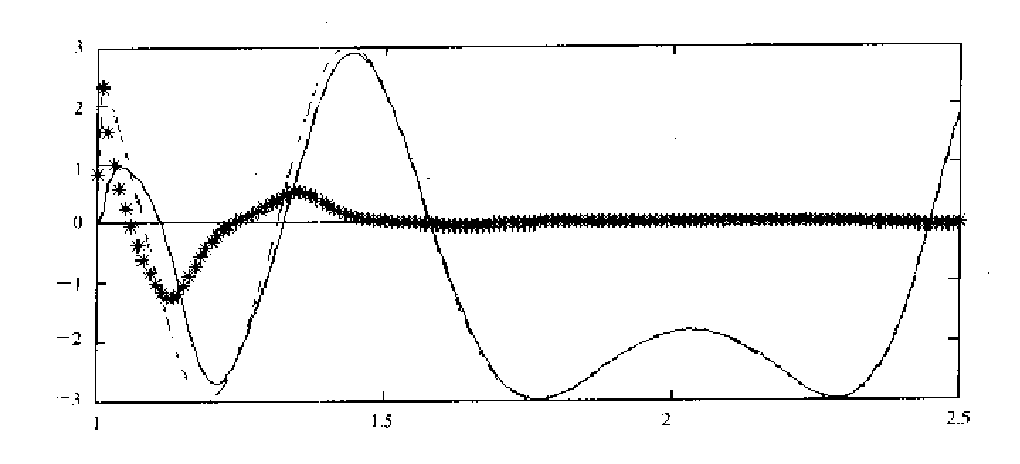



图 12.16　神经网络自适应学习过程及结果

模式识别是神经网络应用最早、也最广泛的领域之一，从最早的感知器到文字识别等都是神经网络应用于模式识别的例子。在传统的统计方法中，模式识别是将样本的特征向量与每个模式类别的特征向量进行比较，然后将样本归到离其最近的模式类别中。而利用神经网络来进行模式识别，不仅可以根据样本进行学习，以改善识别能力，而且不需要对模式分布进行一些统计上的先验假设，以提高其自适应性。

通过分析人的模式识别过程可以看出，对物体的识别包括两个步骤：一是特征分析，即在大脑中寻找一个与该特征相匹配的模式；二是决策分类，经过分析后判断该物体属于哪一个集合类别。计算机模式识别也要遵从这两个步骤进行。人作为老师要对机器进行训练，使机器能够学会分析物体的特征并进行分类，计算机的这种能力称为学习。例如，使用以下方法来训练计算机，使其能够进行直线交角的分析。给定两条直线 $AB$、$AC$，首先要根据点 $A$ 和 $B$ 的坐标来确定直线 $AB$ 的斜率

$$k_{ab}=\frac{y_b-y_a}{x_b-x_a}$$

然后计算 $AB$ 与 x 轴的夹角

$$\alpha_{ab}=\arctan k_{ab}$$

同样计算 $AC$ 与 x 轴的夹角：

$$\alpha_{ac} = \arctan\frac{y_c - y_a}{x_c - x_a}$$

最后得出两条直线的交角：

$$\alpha = \alpha_{ac} - \alpha_{ab}$$

根据交角结果判断直线交角类型：如果 $\alpha > 90°$，则交角为钝角；如果 $\alpha < 90°$，则交角为锐角；否则为直角。通过这几个步骤，计算机就学会了如何判断两条直线的交角类型。

BP 网络是一种比较适合于模式识别分类的网络类型。利用 BP 网络对分类器进行训练，通过调整中间层的权系数最终使输入与某个输出目标一致，实现对对象的分类。相对于其他方法而言，利用神经网络来解决模式识别问题有三点优势：一是神经网络对问题的先验知识要求较少；二是可以实现对特征空间较为复杂的划分；三是它适合用高速并行处理系统来实现。人脑具有令人惊讶的模式识别能力，那么这也就意味着人工神经网络很可能最终也具有较强的识别能力。但是限于神经网络技术的发展，目前由人工神经网络实现的模式识别只能达到一般统计分类器的性能水平。

## 【习　题】

1. 了解小波级数和小波选取等有关技术。

2. 收集有关分形技术的资料，尝试编写一个用于形状分析的 MATLAB 函数。

3. 通过在线帮助了解 MATLAB 所支持的几种神经网络类型，如 BP 网络、Hopfield 网络、最速下降后向网络等。

# 附　录　MATLAB 图像处理工具箱函数集

## 图像显示

colorbar：显示颜色条
getimage：从坐标轴获取图像数据
image：创建并显示图像对象
immovie：根据多帧索引图像创建电影
imshow：显示图像
imagesc：标度数据并显示为图像
montage：以矩形蒙太奇形式显示多个图像帧
subimage：在一个单独图形窗口中显示多帧图像
truesize：调节图像显示大小
warp：将图像显示为纹理表面
zoom：缩放二维图形或图像

## 图像文件 I/O

dicominfo：从 DICOM 消息中读取图元数据
dicomread：读取 DICOM 消息
imfinfo：返回图像文件有关信息
imread：读取图像文件
imwrite：写图像文件

## 空间变换

checkerboard：创建国际象棋盘图像
findbounds：为空间变换寻找输出范围
fliptform：区分 TFORM 结构中的输入和输出角色
imcrop：剪切图像
imresize：重置图像大小
imrotate：旋转图像
interp2：二维图像插值
imtransform：对图像应用二维空间变换
makeresampler：创建重采样结构
maketform：创建几何变换结构

tformarray：多维数组的几何变换
tformfwd：使用前向几何变换
tforminv：使用逆几何变换

## 像素值与统计

corr2：计算二维相关系数
imcontour：创建图像数据的等高线图
imfeature：计算图像区域的特征度量
imhist：显示图像数据的直方图
impixel：判断像素颜色值
improfile：计算线段上的像素值交叉项
mean2：计算矩阵元素的平均值
pixval：显示有关图像像素的信息
regionprops：度量图像区域属性
std2：计算矩阵元素的标准偏差

## 图像分析

edge：寻找灰度图像边界
qtdecomp：实现四叉树分解
qtgetblk：获取四叉树分解中块的数值
qtsetblk：设置四叉树分解中块的数值

## 图像算术

imabsdiff：计算两幅图像的绝对偏差
imadd：两幅图像相加，或给图像加上一个常数
imcomplement：补足图像
imdivide：两幅图像相除，或使图像除以一个常数
imlincomb：计算图像的线性联合
immultiple：两幅图像相乘，或使图像乘以一个常数
imsubtract：两幅图像相减，或使图像减去一个常数

## 图像增强

histeq：使用直方图均一化方法增强对比度
imadjust：调节图像灰度值或调色板
imnoise：添加图像噪声
medfilt2：实现二维中值滤波
ordfilt2：实现二维统计滤波
stretchlim：寻找扩展图像的对比度范围
wiener2：实现二维自适应噪声消除滤波

## 图像匹配

cpcorr：使用互相关性调整控制点位置
cp2tform：根据控制点对推断几何变换
cpselect：控制点选择工具
cpstruct2pairs：将 CPSTRUCT 转换为有效的控制点对
normxcorr2：规格化二维互相关性

## 线性滤波

conv2：实现二维卷积
convmtx2：计算二维卷积矩阵
convn：实现 N－D 卷积
filter2：实现二维滤波
fspecial：创建预定义滤波器
imfilter：多维图像滤波

## 二维线性滤波器设计

freqspace：判断二维频率响应间隔
freqz2：计算二维频率响应
fsamp2：使用频率采样设计二维 FIR 滤波
ftrans2：使用频率变换设计二维 FIR 滤波
fwind1：使用一维窗口方法设计二维 FIR 滤波
fwind2：使用二维窗口方法设计二维 FIR 滤波

## 图像变换

dct2：计算二维离散余弦变换
dctmtx：计算离散余弦变换矩阵
fft2：计算二维快速傅立叶变换
fftn：计算 N－D 快速傅立叶变换
fftshift：逆转 FFT 输出的四个象限
idct2：计算二维逆离散余弦变换
ifft2：计算二维逆快速傅立叶变换
ifftn：计算 N－D 逆快速傅立叶变换
iradon：计算逆 Radon 变换
phantom：生成头幻影图像
radon：计算 Radon 变换

## 邻域和块操作

bestblk：选择块处理的大小

blkproc：实现图像的离散块处理
col2im：对矩阵列进行重新排列成块
colfilt：使用列风格函数进行邻域操作
im2col：对块进行重新排列成为矩阵列
nlfilter：实现一般的滑动邻域操作

## 形态操作(灰度和二进制图像)

conndef：缺省的连通性数组
imbothat：实现低帽滤波
imclearborder：抑制连接到图像边界的光结构
imclose：图像形态关闭
imdilate：膨胀图像
imerode：腐蚀图像
imextendedmax：扩展极大值变换
imextendedmin：扩展极小值变换
imfill：填充图像区域
imhmax：H 极大值变换
imhmin：H 极小值变换
imimposemin：强调极小值
imopen：图像形态开启
imreconstruct：实现形态重建
imregionalmax：图像区域极大值
imregionalmin：图像区域极小值
imtophat：实现高帽滤波
watershed：寻找图像分水岭区域

## 形态操作(二进制图像)

applylut：使用查表方法实现邻域操作
bwarea：计算二进制图像中的对象面积
bwareaopen：二进制图像区域打开，消除小物体
bwdist：距离变换
bweuler：计算二进制图像的欧拉数
bwfill：填充二进制图像的背景区域
bwhitmiss：二进制避碰操作
bwlabel：标记二维二进制图像连接成分
bwlabeln：标记 N－D 二进制图像连接成分
bwmorph：对二进制图像执行形态操作
bwpack：对二进制图像打包
bwperim：寻找二进制图像中的主要对象

bwselect：选择二进制图像中的对象
bwulterode：最终腐蚀
bwunpack：对打包的二进制图像解包
imregionalmin：计算图像的区域极小值
imtophat：实现高帽滤波
makelut：构造 applylut 所使用的表

## 结构元素的创建与操作

getheight：获取结构元素的高度
getneighbors：获取结构元素的相邻位置和高度
getnhood：获取结构元素邻域
getsequence：从分解的结构元素中提取序列
isflat：对于平结构元素则返回 true 值
reflect：反射结构元素
strel：创建形态结构元素
translate：变换结构元素

## 去模糊

deconvblind：使用盲去卷积重建图像
deconvlucy：使用加速 Richardson-Lucy 算法重建图像
decongvreg：使用规则化滤波器重建图像
deconvwnr：使用维纳滤波器重建图像
edgetaper：使图像边缘逐渐衰弱
otf2psf：将光学转移函数变换为点扩散函数
psf2otf：将点扩散函数变换为光学转移函数

## 数组操作

circshift：循环切换数组
padarray：填充数组

## 基于区域的操作

roicolor：根据颜色选择感兴趣的区域
roifill：在任意区域内进行光滑插值
roifilt2：对感兴趣区域滤波
roipoly：选择感兴趣的多边形区域

## 调色板操作

brighten：使调色板变亮或变暗
cmpermute：重新排列调色板中的颜色

cmunique：寻找惟一的调色板颜色和相应的图像
colormap：设置或获取颜色列表
imapprox：使用一幅颜色较少的图像来近似索引图像
rgbplot：绘制 RGB 调色板成分

## 调色板空间变换

hsv2rgb：将 HSV 值转换为 RGB 颜色间隔
ntsc2rgb：将 NTSC 值转换为 RGB 颜色间隔
rgb2hsv：将 RGB 值转换为 HSV 颜色间隔
rgb2ntsc：将 RGB 值转换为 NTSC 颜色间隔
rgb2ycbcr：将 RGB 值转换为 YCbCr 颜色间隔
ycbcr2rgb：将 YCbCr 值转换为 RGB 颜色间隔

## 图像类型和类型转换

dither：使用抖动方法转换图像
double：将数据转换为双精度类型
gray2ind：将灰度图像转换为索引图像
grayslice：使用阈值根据灰度图像创建索引图像
graythresh：使用 Otsu's 方法计算全局图像阈值
im2bw：使用阈值将图像转换为二进制图像
im2double：将图像数组转换为双精度类型
im2mis：将图像转换为 Java 图像
im2uint16：将图像数组转换为 16 位无符号整数
im2uint8：将图像数组转换为 8 位无符号整数
ind2gray：将索引图像转换为灰度图像
ind2rgb：将索引图像转换为 RGB 图像
isbw：如果是二进制图像，则返回 true 值
isgray：如果是灰度图像，则返回 true 值
isind：如果是索引图像，则返回 true 值
isrgb：如果是 RGB 图像，则返回 true 值
label2rgb：将标记矩阵转换为 RGB 图像
mat2gray：将矩阵转换为灰度图像
rgb2gray：将 RGB 图像或调色板转换为灰度图像
rgb2ind：将 RGB 图像转换为索引图像
uint16：将数据转换为 16 位无符号整数
uint8：将数据转换为 8 位无符号整数

# 各章习题答案与提示

**第一章**

1. 68.27 秒；1092.27 秒

2. 调用以下语句：

```
bitmap = imread('mybitmap.bmp', 'bmp');
imwrite(bitmap, 'mybitmap.png', 'jpg');
```

**第二章**

1. 略。

2. 假设地图图像为 worldmap.tif，调用以下语句：

```
[x, y, z] = sphere;
I = imread('worldmap.tif');
warp(x, y, z, I);
```

**第三章**

1. 调用以下代码：

```
I=imread('rice.tif');
X=ones(size(I));
I1=im2double(I);
J1=I1.*I1-X/256;
J2=I1*0.6+X/64;
subplot(1, 2, 1), imshow(J);
subplot(1, 2, 2), imshow(J2);
```

2. 调用以下代码：

```
I=imread('rice.tif');
[m, n]=size(I);
I1=zeros(m, n);
I1(1:m, 1:n-10)=I(1:m, 11:n);
I1=uint8(I1);
sub=imsubtract(I1, I);
div=imdivide(I1, I);
subplot(1, 2, 1), imshow(sub);
subplot(1, 2, 2), imshow(div, []);
```

3. 调用以下代码：

```
I = imread('tire.tif');
f = inline('uint8(round(mean2(x) * ones(size(x))))');
I2 = colfilt(I, [3 3], 'distinct', @f);
```

4. (略)

**第四章**

1. 绘制三幅相同大小的图像，假设分别命名为 exe1.tif、exe2.tif 和 exe3.tif。调用以

下代码：

```
I1=imread('exe1.tif');
I2=imread('exe2.tif');
I3=imread('exe3.tif');
s=1;
[m n]=size(I1);
for k=1:100
    s=s*2;
    if(and(m<s, n<s)) break; end;
end
F1 = fft2(I1, s, s);
F12 = fftshift(F1);
F2 = fft2(I2, s, s);
F22 = fftshift(F2);
F3 = fft2(I3, s, s);
F32 = fftshift(F3);
subplot(1, 3, 1), imshow(log(abs(F12)), [-1 5]); colormap(jet)
subplot(1, 3, 2), imshow(log(abs(F22)), [-1 5]); colormap(jet)
subplot(1, 3, 3), imshow(log(abs(F32)), [-1 5]); colormap(jet)
```

2. 调用以下代码：

```
I = imread('cameraman.tif');
I = im2double(I);
T = dctmtx(8);
B = blkproc(I, [8 8], 'P1*x*P2', T, T');
mask = [1 1 1 1 1 1 0 0
        1 1 1 1 1 0 0 0
        1 1 1 1 0 0 0 0
        1 1 1 0 0 0 0 0
        1 1 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0];
B2 = blkproc(B, [8 8], 'P1.*x', mask);
I2 = blkproc(B2, [8 8], 'P1*x*P2', T', T);
subplot(1, 2, 1), imshow(I)
subplot(1, 2, 2), imshow(I2)
figure, imshow(imsubtract(I, I2)), colormap(jet);
```

3. 假设用 $b_i$ 代替略去的分量 $y_i$，则 $\boldsymbol{X}$ 的估计值为

$$\hat{X} = \sum_{i=1}^{m} y_i A_i + \sum_{i=m+1}^{n^2} b_i A_i$$

产生的误差为

$$\varepsilon = \sum_{i=m+1}^{n^2} (y_i - b_i) A_i$$

可以推出均方差值为

$$E\{\|\varepsilon\|^2\} = E\{\varepsilon^T\varepsilon\} = \sum_{i=m+1}^{n^2} E\{(y_i - b_i)^2\}$$

令

$$\frac{\partial}{\partial b_i}E\{\|\varepsilon\|^2\} = 0$$

得最佳略去分量即略去分量的平均值：

$$b_i = E(y_i) = \overline{y_i} \qquad m+1 < i < n^2$$

由此可以推出：

$$E\{\|\varepsilon\|^2\} = \sum_{i=m+1}^{n^2} A_i^T \Sigma_x A_i$$

用拉格朗日乘法求最佳的 $A_i$，使上式最小，并满足条件 $A_i^T A_i = 1$，可得：

$$E\{\|\varepsilon\|^2\} = \sum_{i=m+1}^{n^2} \lambda_i$$

其中，$\lambda_i$ 为 $\Sigma_x$ 的特征值。由此可见，如果将与最小的 $n^2 - m$ 个 $\lambda_i$ 相对应的 $y_i$ 略去，那么就会得到最小的均方误差。证毕。

**第五章**

1. 交换律：对 $f * g = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\mathrm{d}\tau$ 进行变量替换 $x = t - \tau$ 得：

$$f * g = \int_{-\infty}^{\infty} f(t-x)g(x)\mathrm{d}x = g * f$$

（其他略）

2. 调用以下代码：

```
[f1, f2] = freqspace(25, 'meshgrid');
Hd = ones(25);
d = exp(0-f1. * f2/0.5);
Hd(d<0.6) = 0;
win = fspecial('gaussian', 25, 0.5);
win = win ./ max(win(:));            % 使最大的窗口值为 1
h = fwind2(Hd, win);
Hd1=zeros(25);
Hd1(d>0.4)=1;
h1=fwind3(Hd1, win);
subplot(1, 2, 1), freqz(h);
subplot(1, 2, 2), freqz(h1);
```

**第六章**

1. 由于洞填充是一种常见操作，所以 imfill 有一种特殊的语法格式支持二进制和灰度图像的洞填充。在这种格式下，可以指定 imfill 函数的可选参数 holes，此时无需为每一个洞指定起始点。代码调用结果如下：

```
blood = imread('blood1.tif');
subplot(1, 2, 1), imshow(blood)
blood2 = imcomplement(imfill(imcomplement(blood), 'holes'));
subplot(1, 2, 2), imshow(blood2)
```

2.（略）

**第七章**

1. 调用以下代码：

```
C = imread('greens.jpg');
D = imtransform( C, ring, 'cubic', ...
                 'UData', uData, 'VData', vData, ...
                 'XData', [-2 2], 'YData', [-2 2], ...
                 'Size', [400 400], 'FillValues', 255 );
```

2. 调用以下代码：

```
truesizewarning = iptgetpref('TruesizeWarning');
iptsetpref('TruesizeWarning', 'off'); % Turn warning off
load mri;
M1 = D(:, 64, :, :); size(M1)
M2 = reshape(M1, [128 27]); size(M2)
T0 = maketform('affine', [0 -2.5; 1 0; 0 0]);
imtransform(M2, T0, 'cubic')
R2 = makeresampler({'cubic', 'nearest'}, 'fill');
M3 = imtransform(M2, T0, R2);
T1 = maketform('affine', [-2.5 0; 0 1; 68.5 0]);
T2 = maketform('custom', 3, 2, [], @ipex003, 64);
Tc = maketform('composite', T1, T2);
R3 = makeresampler({'cubic', 'nearest', 'nearest'}, 'fill');
M4 = tformarray(D, Tc, R3, [4 1 2], [1 2], [66 128], [], 0);
T3 = maketform('affine', [-2.5 0 0; 0 1 0; 0 0 0.5; 68.5 0 -14]);
S = tformarray(D, T3, R3, [4 1 2], [1 2 4], [66 128 35], [], 0);
figure;
immovie(S, map);
S2 = padarray(S, [6 0 0 0], 0, 'both');
montage(S2, map);
function U = ipex003( X, t )
U = [X repmat(t.tdata, [size(X, 1) 1])];
```

**第八章**

1. 调用 blkproc 函数实现(略)

2. 调用以下代码：

```
b = remez(10, [0 0.4 0.5 1], [1 1 0 0]);          %一维最优波纹滤波器设计
h = ftrans2(b);
I=imread('rose.tif');
I= imfilter(I, h);
```

**第九章**

(略)

**第十章**

1. 首先将图像的行、列调换：

```
[m n]=size(I)
for k=1:m
    for l=1:n
        J(k, l)=I(l, k)
    end
end
```

然后调用介绍的行程编码方法即可。

2. 首先使用与哈夫曼编码相同的方法，求出 $m\times n$ 图像中存在的颜色总数 $s$ 及出现概率，并存放在数组 $P[1:s, 2]$中，然后调用以下代码：

```
len=1;
for k=1:m
    for l=1:n
        for n=1:s
            if(I(k, l)==P(n, 1))
                len=len * P(n, 2);
                break;
            end
        end
    end
end
```

最终得到的 len 即为码字。

3. 假设方程为 AX=B，则调用以下语句即可求出 X：

```
X=A\B
```

**第十一章**

1. 调用以下代码：

```
I = imread('cell.tif');
DI = imadjust(I, [], [0 1]);
BWs = edge(DI, 'sobel', (graythresh(DI) * .1));
se90 = strel('line', 3, 90);
se0 = strel('line', 3, 0);
BWsdil = imdilate(BWs, [se90 se0]);
BWdfill = imfill(BWsdil, 'holes');
BWnobord = imclearborder(BWdfill, 4);
seD = strel('diamond', 1);
BWfinal = imerode(BWnobord, seD);
BWfinal = imerode(BWfinal, seD);
BWoutline = bwperim(BWfinal);
Segout = imadd(I, immultiply(BWoutline, 255));
```

2. (略)

**第十二章**

(略)

# 参 考 文 献

[1] 章毓晋. 图像工程(上册)图像处理和分析. 北京：清华大学出版社，1999
[2] 张远鹏，董海，周文灵. 计算机图像处理技术基础. 北京：北京大学出版社，1996
[3] 阮秋琦编. 数字图像处理基础. 北京：中国铁道出版社，1988
[4] [美]A. 帕普力斯著. 数字图像处理学. 谢国瑞等译. 北京：科学出版社，1984
[5] 崔锦泰. 程正兴译. 小波分析导论. 西安：西安交通大学出版社，1995
[6] 潘志编著. 近代分析数学应用基础. 徐州：中国矿业大学出版社，1993
[7] 李介谷，蔡国廉等编. 计算机模式识别技术. 上海：上海交通大学出版社，1986
[8] 沈清，胡德文，时春编著. 神经网络应用技术. 长沙：国防科技大学出版社，1993
[9] [美]K. R. Castleman 著. 数字图像处理. 朱志刚等译. 北京：电子工业出版社，1998
[10] 朱志刚. 数字图像处理. 内部讲义. 清华大学计算机系，1998
[11] 崔屹. 数字图像处理技术与应用. 北京：电子工业出版社，1997
[12] 吕风军. 数字图像处理编程入门. 北京：清华大学出版社，1999